Hybridizing metaheuristics with machine learning for combinatorial optimization : a taxonomy and learning to select operators

Maryam Karimi Mamaghan

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Maryam KARIMI MAMAGHAN

Hybridizing Metaheuristics with Machine Learning for Combinatorial Optimization: A Taxonomy and Learning to Select Operators

Thèse présentée et soutenue à Brest, le 20 juillet 2022
Unité de recherche : Lab-STICC/DECIDE
Thèse N° : 2022IMTA0297

Rapporteurs avant soutenance :

Kenneth SORRENSEN	Professeur, University of Antwerp, Anvers, Belgique
Jin-Kao HAO	Professeur, Université d'Angers, Angers, France

Composition du Jury :

Président :	Olga BATTAIA	Professeur, KEDGE Business School, Bordeaux, France
Examinateurs :	Kenneth SORRENSEN	Professeur, University of Antwerp, Anvers, Belgique
	Jin-Kao HAO	Professeur, Université d'Angers, Angers, France
	Sébastien VEREL	Professeur, Université du Littoral Côté d'Opale, Dunkerque, France
	Axel PARMENTIER	Maître de Conférences, École des Ponts ParisTech, Paris, France
	Bastien PASDELOUP	Maître de Conférences, IMT Atlantique, Brest, France
Dir. de thèse :	Patrick MEYER	Professeur, IMT Atlantique, Brest, France

Invité :
Mehrdad MOHAMMADI — Maître de Conférences, IMT Atlantique, Brest, France

Acknowledgments

I would like to first thank my supervisory team, Patrick Meyer, Mehrdad Mohammadi, and Bastien Pasdeloup for their support during these three years. It was a great experience working with them.

I would like to express my deepest gratitude to my jury members, Kenneth Sorrensen, Jin-kao Hao, Olga Battaia, Sebastien Verel, and Axel Parmentier for their evaluation and valuable remarks on my work and their presence in my PhD defense.

I am grateful to Ender Ozcan, who invited me for a research visit at the University of Nottingham, for his support and sharing knowledge throughout our collaboration, and all the members of COL lab at the University of Nottingham who made good moments for me during my stay at Nottingham. It was really nice to meet them all and have interesting discussion with most of them.

I am also thankful to Nadia Lahrichi and Andrea Lodi who invited me for a research visit at Polytechnique Montreal for their valuable remarks and exchange during our collaboration.

I would like to thank Ecole des Docteurs Bretagne Loire, PÃŧle Doctoral de Rennes, and Ecole Doctoral MathSTIC, and IMT ATlantique Bretagne-Pays de la Loire for their financial support that made it possible for me to have two research visit during my PhD. IâĂŹd like to acknowledge my colleagues in DECIDE team and the Lab-STICC research lab with whom I had nice discussions through the monthly seminars of the team.

Words cannot express my gratitude to my family for their unconditional support during pursuing my PhD.

I could not have undertaken this journey without the support and love of my dearest husband, who shared this entire journey with me with all its ups and downs. Thank you for being an inspiration to me.

Contents

Acknowledgments **3**

Résumé **15**

Abstract **25**

1 Introduction **27**
1.1 Introduction . . . **28**
1.2 Contribution of thesis . . . **30**
1.2.1 ML-into-MH: A taxonomy . . . 30
1.2.2 ML-into-MH: Learning to select operators . . . 31
1.3 Thesis outline . . . **32**

2 Theoretical Background **33**
2.1 Combinatorial optimization problems . . . **34**
2.1.1 Traveling salesman problem . . . 34
2.1.2 Permutation flowshop scheduling problem . . . 35
2.2 Meta-heuristics . . . **36**
2.3 Machine learning . . . **39**
2.3.1 Supervised learning algorithms . . . 39
2.3.2 Unsupervised learning algorithms . . . 40
2.3.3 Semi-supervised learning . . . 41
2.3.4 Reinforcement learning algorithms . . . 41
2.4 Adaptive Operator Selection & Learning . . . **43**

3 Integration of Machine Learning into Meta-heuristics: A Taxonomy **45**
3.1 Taxonomy and review methodology . . . **46**
3.1.1 Contributions . . . 46
3.1.2 Taxonomy . . . 47
3.1.3 Search methodology . . . 48
3.2 Algorithm selection . . . **49**
3.2.1 Literature classification & analysis . . . 52
3.2.2 Discussion & future research directions . . . 54
3.2.2.1 Guideline & requirements . . . 54
3.2.2.2 Challenges & future research directions . . . 54
3.3 Fitness evaluation . . . **56**
3.3.1 Literature classification & analysis . . . 57
3.3.2 Discussion & future research directions . . . 58

3.4 Initialization **59**
3.4.1 Literature classification & analysis 60
3.4.2 Discussion & future research directions 61
3.5 Evolution **62**
3.5.1 Operator selection 62
3.5.1.1 Literature classification & analysis 66
3.5.1.2 Discussion & future research directions 69
3.5.2 Learnable evolution model 71
3.5.2.1 Literature classification & analysis 72
3.5.2.2 Discussion & future research directions 73
3.5.3 Neighbor generation 74
3.5.3.1 Literature classification & analysis 74
3.5.3.2 Discussion & future research directions 75
3.6 Parameter setting **76**
3.6.1 Literature classification & analysis 77
3.6.2 Discussion & future research directions 78
3.7 Cooperation **79**
3.7.1 Literature classification & analysis 80
3.7.2 Discussion & future research directions 82
3.8 Conclusion **82**

4 Q-learning for Operator Selection: A General Framework 85
4.1 General framework **86**
4.2 Application to TSP **89**
4.2.1 Set of search operators 89
4.2.2 QILS framework to select local search operators 93
4.2.3 QILS framework to select perturbation operators 95
4.3 Application to PFSP **95**
4.3.1 Set of search operators 96
4.3.2 QILS framework to select perturbation operators 97
4.4 Experimental design **99**
4.4.1 Dataset 99
4.4.2 Benchmark comparison 99
4.4.3 Performance comparison metrics 101
4.4.4 Statistical test 101
4.4.5 Parameters tuning 102
4.5 Computational results **104**
4.5.1 Application of QILS to TSP 104
4.5.2 Application of the QILS framework to PFSP 107
4.5.2.1 Phase 1: comparison between QILS, RILS, and IILS algorithms 107
4.5.2.2 Phase 2: comparison of QILS with state-of-the-art algorithms 113
4.5.2.3 Adaptiveness of operators to the problem instances . . . 122
4.5.2.4 Sensitivity analysis 124
4.5.3 Complexity analysis 129

5 Conclusions and Future Research 131

Publications 139

A Appendices **141**
A.1 List of COPs **141**
A.2 Best-known solutions for PFSP **143**

Bibliography **147**

List of Figures

1 Cadre QILS proposé 21

2 Iterated local search algorithm 38
3 A generic scheme of RL algorithm 41
4 AOS with an online learning 44

5 Taxonomy on the use of ML in MHs (ML-in-MH) 48
6 Number of papers per year and per each type of integration of ML techniques in MHs for solving COPs 50
7 Procedure of ASP 51
8 Exploration vs. Exploitation of selection methods 68
9 Cooperation procedure 80

10 Flowchart of the proposed QILS framework 87
11 Independent improving moves in the *best-independent-moves 2-opt* operator 91
12 Performance comparison of $IILS_d$ based on ARPD (%) for *Taillard* dataset 103
13 Boxplot of QILS, RILS and $IILS_d$ based on RPD (%) for each instance set of *Taillard* dataset. The average performance of each algorithm is shown using red circles, and the median value is shown on the right side of each boxplot. 110
14 Boxplot of QILS, RILS and $IILS_d$ based on CPU time (s) for each instance set of *Taillard* dataset 111
15 Convergence rate of QILS, RILS and $IILS_d$ for each instance set of *Taillard* dataset 112
16 Boxplot of QILS, RILS and $IILS_d$ based on normalized objective function value over all instances of each dataset 113
17 Boxplot of QILS, RILS and $IILS_d$ based on normalized CPU time (s) over all instances of each dataset 113
18 Boxplot of QILS, RILS, and benchmark algorithms based on RPD (%) for scale $t = 120$ for each instance set of *Taillard* dataset 121
19 Boxplot of QILS and benchmark IGs based on normalized objective value over all instances of each dataset 122
20 Relative improvement index for each perturbation operator in certain instance sets of *Taillard* dataset 123
21 Application of each perturbation operator at each step of the search process in certain instances of *Taillard* dataset 124
22 Sensitivity of QILS to its parameters. Values found in Table 13 are highlighted in red. 125
23 Performance comparison of QILS and RILS based on ARPD (%) for different sizes of the action set on *Taillard* dataset 126

List of Tables

1 Number of Hamiltonian tours for TSP 28

2 Classification of papers studying ASP 52
3 Classification of papers studying fitness evaluation 58
4 Classification of papers studying initialization 61
5 Credit assignment methods 64
6 Selection methods 65
7 Move acceptance methods 66
8 Classification of papers studying AOS 67
9 Classification of papers studying LEM 73
10 Classification of papers studying neighbor generation 75
11 Classification of papers studying parameter setting 77
12 Classification of papers studying cooperation 81

13 Parameters of QILS, their corresponding levels and tuned values 103
14 Performance comparison of QILS for local search operator selection with the corresponding RILS, and IILS in TSP application. The best values for each set of instances among different algorithms have been highlighted in gray. Furthermore, bold values indicate results that are not statistically distinguishable from results of the QILS algorithm. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.003, 0.072)$. 105
15 Performance comparison of QILS for perturbation operator selection with the corresponding RILS, and IILS for TSP application. The best values for each set of instances among different algorithms have been highlighted in gray. Furthermore, bold values indicate results that are not statistically distinguishable from results of the QILS algorithm. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.001, 0.048)$. 106
16 Performance comparison of QILS, RILS and IILS_d on *Taillard* dataset. The best values for each set of instances among different algorithms have been highlighted in gray. Furthermore, bold values indicate results that are not statistically distinguishable from results of the QILS algorithm. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.004, 0.041)$. 108
17 Performance comparison of QILS, RILS and IILS_d on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.005, 0.040)$. 108
18 Performance comparison of QILS, RILS and IILS_d on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.001, 0.037)$. 109

19 ARPD of QILS, RILS, and IILS$_d$ for different datasets 112
20 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 60$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.004, 0.047)$. . . . 115
21 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 90$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.003, 0.019)$. . . . 115
22 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 120$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.035)$. . 116
23 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 60$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.017, 0.048)$. 116
24 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 90$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.011, 0.042)$. 117
25 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 120$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.008, 0.049)$. 118
26 Results of comparing QILS with RILS and algorithms for scale $t = 60$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.047)$. . . . 118
27 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 90$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.049)$. 119
28 Results of comparing QILS with RILS and benchmark algorithms for scale $t = 120$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.049)$. 120
29 ARPD of QILS, RILS, and benchmark algorithms for different datasets . 122
30 Results of comparing QILS with the proposed local/global reward mechanism to QILS with 0/1 reward mechanism for scale $t = 120$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.013, 0.044)$. 126
31 Performance comparison of QILS and RILS for different actions on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.004, 0.042)$. 127
32 Complexity of QILS 130

A.1 Exhaustive list and the abbreviation (Abv.) of the COPs studied by the articles reviewed in this thesis 141
A.2 Best-known solutions (C^*_{max}) vs. the best C_{max} found by QILS for each instance over 30 runs for time scale $t = 120$ of *Taillard* dataset. LB and UB stand for lower bound and upper bound, respectively. 143
A.3 Best-known solutions (C^*_{max}) vs. the best C_{max} found by QILS for each instance over 30 runs for time scale $t = 120$ of *VRF-hard-small* dataset. LB and UB stand for lower bound and upper bound, respectively. 144

A.4 Best-known solutions (C^*_{max}) vs. the best C_{max} found by QILS for each instance over 30 runs for time scale $t = 120$ of *VRF-hard-large* dataset. LB and UB stand for lower bound and upper bound, respectively. 145

Résumé

Les problèmes d'optimisation combinatoire (POC) sont une classe de problèmes d'optimisation difficiles à résoudre. Le mot combinatoire concerne la sélection, l'arrangement ou le séquençage d'une collection d'objets. L'optimisation combinatoire est la recherche de la meilleure sélection, disposition ou séquence par rapport à une fonction objectif dans un espace de recherche fini. Il existe une variété de types de POC qui ont été largement étudiés par la communauté de recherche opérationnelle. Un POC classique qui a été largement étudié dans différents domaines de recherche est le problème du voyageur de commerce (Traveling Salesman Problem - TSP en anglais). Dans le TSP, il y a un vendeur qui doit effectuer une tournée en visitant un ensemble de n villes exactement une fois et en revenant au point de départ (c'est-à-dire une tournée hamiltonienne). Chaque arc reliant deux villes i et j est associé à une distance d_{ij} et le but est de trouver le tour hamiltonien des villes avec une distance totale minimale. Considérons un algorithme de recherche exhaustive qui évalue tous les tours hamiltoniens possibles d'une instance de TSP donnée et renvoie le tour avec la distance totale minimale comme la solution optimale. Lorsque le nombre de villes est faible (par exemple, $n \leq 6$), un algorithme de recherche exhaustif est efficace et trouve rapidement la solution optimale. Cependant, à mesure que le nombre de villes augmente, ce qui est le cas dans les problèmes du monde réel, il devient de plus en plus fastidieux (voire impossible à calculer) d'utiliser un algorithme de recherche exhaustif pour trouver la solution optimale pour le TSP.

Pour être plus précis, le nombre de tours hamiltoniens possibles pour une instance de TSP avec n villes avec une matrice de distance symétrique entre les villes vaut $(n-1)!/2$. En conséquence, si le nombre de villes augmente même de petites valeurs, le nombre de tours hamiltoniens augmente d'un ordre de $(n-1)!/2$. Cela implique que la complexité de calcul d'un algorithme de recherche exhaustive est $\mathcal{O}(n!)$ pour résoudre une instance de TSP de taille n. Par exemple, pour une instance de TSP de petite taille avec un nombre $n = 20$ de villes, un algorithme de recherche exhaustif devrait évaluer $6,08006E+16$ tours hamiltoniens différents. Considérant que chaque évaluation ne prend qu'une μs, une instance de TSP avec $n = 20$ villes prend environ vingt mille siècles pour être résolue de manière exhaustive.

En général, le TSP est classé comme une classe NP-difficile de problèmes d'optimisation pour lesquels aucun algorithme de résolution connu avec une complexité polynomiale en temps de $\mathcal{O}(Cn^k)$ (pour certains k et $C > 0$) n'existe [Woe03]. Semblable au TSP, la plupart des POC appartiennent à la classe NP-difficile des problèmes d'optimisation. Tout problème NP-difficile peut être résolu en utilisant une recherche exhaustive. Cependant, lorsque la taille des instances augmente, le temps d'exécution d'une recherche exhaustive devient extrêmement élevé, même pour des instances de taille relativement petite dans le monde réel.

Les algorithmes de recherche exhaustive appartiennent à une classe générale d'algorithmes appelés algorithmes *exacts* [LN87 ; Woe03]. Les algorithmes exacts (par exemple,

branch-and-bound, décomposition de Benders, etc.), comme leur nom l'indique, sont des algorithmes qui résolvent toujours un problème d'optimisation de manière optimale [FK13]. Résoudre les instances de TSP du monde réel comprenant des milliers de villes à l'optimalité serait très difficile et prendrait beaucoup de temps (voire impossible à calculer).

La complexité inhérente aux POC rend nécessaire l'utilisation d'algorithmes approchés. Les algorithmes approchés sont une classe d'algorithmes d'optimisation qui sont principalement développés pour résoudre des problèmes d'optimisation NP-difficiles [GK06]. Ces algorithmes sont capables de trouver des solutions (quasi) optimales en un temps de calcul raisonnable ; cependant, ils ne garantissent pas l'optimalité des solutions obtenues. Parmi les différents algorithmes d'approchés, les algorithmes *Méta-heuristiques* (MH) sont des paradigmes d'intelligence computationnelle largement utilisés pour résoudre des problèmes d'optimisation complexes, en particulier (POC), s'ils sont bien conçus et adaptés [OL96]. Compte tenu de leur capacité à trouver des solutions (quasi) optimales pour les POC en un temps de calcul raisonnable, ils sont de bons substituts aux algorithmes exacts [HW90 ; OL96 ; Tal09]. C'est la raison principale de la croissance significative de l'intérêt pour la conception et l'emploi des MHs au cours des dernières décennies.

D'un point de vue technique, les MHs sont des algorithmes d'optimisation qui organisent et pilotent une interaction entre des procédures d'amélioration locale et des stratégies globales afin de créer un processus de recherche itératif capable d'échapper aux optima locaux et d'effectuer une recherche robuste d'un espace de recherche [GP10]. Au cours d'un tel processus de recherche itératif, un nombre considérable de solutions sont générées, évaluées et évoluent jusqu'à ce qu'une solution prometteuse soit obtenue. En fait, pendant le processus de recherche, les MHs génèrent un volume considérable de données, y compris les bonnes (élites) ou mauvaises solutions en termes de leurs valeurs de fitness, la séquence des opérateurs de recherche du début à la fin, les trajectoires d'évolution des différentes solutions, les optima locaux, etc. Ces données sont potentiellement porteuses de connaissances utiles telles que les propriétés des bonnes et mauvaises solutions, la performance des différents opérateurs à différentes étapes du processus de recherche, la préséance des opérateurs de recherche, etc.

Les MHs classiques n'utilisent aucune forme de connaissance cachée dans ces données pour résoudre les POC. Cependant, au cours de la dernière décennie, un grand intérêt a été porté à l'exploitation de ces données et à l'extraction de connaissances utiles pour la résolution des POC. À cet égard, un nombre considérable d'études se sont penchées sur l'utilisation de techniques d'apprentissage automatique (AA) (Machine Learning - ML en anglais) dans la résolution de POC. D'une manière générale, l'AA est un sous-domaine de l'intelligence artificielle qui utilise des approches algorithmiques et statistiques pour donner aux ordinateurs la capacité d'"apprendre" à partir de données, c'est-à-dire d'améliorer leurs performances dans la résolution de tâches sans être explicitement programmés pour chacune d'entre elles. Ces systèmes améliorent leur apprentissage au fil du temps de manière autonome, en utilisant les connaissances extraites des données et des informations sous forme d'observations et d'interactions avec le monde réel. Les connaissances acquises permettent à ces systèmes de se généraliser correctement à de nouveaux contextes.

Les techniques d'AA ont été utilisées pour résoudre les POC de deux manières ; premièrement, pour résoudre directement les POC [JG10], et deuxièmement, être intégré dans les MHs pour extraire des connaissances utiles des données générées et les injecter dans le processus de recherche dans le but d'améliorer les performances des MHs [DLPS17].

L'objectif principal de l'intégration d'AA dans les MHs est d'améliorer l'efficacité des MHs en accélérant le processus de recherche par des approximations, en injectant des connaissances dans le processus de recherche ou en trouvant de meilleures solutions en remplaçant les connaissances et l'expérience des experts par de meilleures décisions obtenues par des techniques d'AA. Plusieurs études ont été réalisées en utilisant différentes techniques d'AA pour différentes tâches dans les MHs. Certaines études se sont concentrées sur l'utilisation de techniques classiques d'AA pour des tâches conventionnelles telles que la génération de solutions initiales ou l'approximation de la fonction de fitness. D'autre part, il existe des études utilisant des techniques d'AA plus avancées pour concevoir des méta-heuristiques plus sophistiquées à travers des cadres coopératifs. Cette intégration peut en effet guider les MHs vers la prise de meilleures décisions et par conséquent rendre les MHs plus intelligentes et améliorer leurs performances en termes de qualité de solution, de taux de convergence et de robustesse.

Au cours de la dernière décennie, l'intégration des techniques d'AA dans les méthodes d'optimisation, en particulier les MHs, a suscité une attention considérable et cette attention croissante a donné lieu à de nombreux articles de recherche. Cependant, il existe encore de nombreuses directions de recherche qui peuvent être approfondies [Tal02 ; STÃ19 ; BLP21b].

Contributions Principales

La contribution de cette thèse au domaine de l'intégration des techniques d'AA dans les MHs (AA-dans-MH) se situe à deux niveaux : *analytique* et *technique*. Au niveau *analytique*, cette thèse fournit, pour la première fois en son genre, une revue complète sur l'intégration AA-dans-MH qui non seulement classifie différents articles de recherche avec différents objectifs d'intégration mais fournit également une analyse complète et une discussion sur les détails techniques de cette intégration (par exemple, les défis, les avantages, les inconvénients, les opportunités, etc.) Au niveau *technique*, nous nous concentrons sur un objectif d'intégration particulier, où nous étudions l'intégration de l'AA pour guider les MHs dans la sélection de l'opérateur de recherche le plus approprié à chaque étape du processus de recherche. Ces contributions sont discutées plus en détail dans les deux sous-sections suivantes.

AA-dans-MH : une taxonomie

À notre connaissance, il n'existe pas de revue complète sur l'intégration de l'AA dans les MHs qui étudie également l'intégration d'un point de vue technique. [JDT06] a fourni une brève étude sur la façon dont les techniques d'AA peuvent aider les MHs, sans discussion détaillée sur la façon dont cette intégration se produit. Une autre étude a été réalisée par [Zha+11] sur la façon dont les techniques d'AA peuvent améliorer les performances des algorithmes de calcul évolutionnaire. En plus, à cet égard, [Tal16] a étudié différentes manières d'hybridation entre différents MHs ainsi que d'hybridation des MHs avec la programmation mathématique, la programmation par contraintes et l'AA. Bien que l'auteur fournisse une bonne vue générale sur la façon d'hybridation des MHs, il est moins concentré sur l'intégration AA-dans-MH.

Plus récemment, des études plus générales et plus complètes ont été réalisées par [STÃ19] et [Tal20] sur l'intégration d'AA et MH. Cependant, [STÃ19] a fourni moins de détails sur l'intégration AA-dans-MH par rapport à [Tal20]. Les deux Ãľtudes manquent une discussion approfondie sur les défis de la recherche et les orientations futures de la recherche pour l'intégration de l'AA dans l'optimisation. Elles n'entrent pas également

dans les détails sur les exigences, les défis, et les possibles directions de recherche futures sur l'utilisation des techniques d'AA dans chaque niveau d'intégration

À notre connaissance, il n'existe aucune étude dans la littérature qui examine de manière complète et technique comment les techniques d'AA peuvent être intégrées dans les MHs et pour quels buts spécifiques. Pour combler ce problème, et après avoir passé en revue les tentatives faites par les communautés de l'informatique et de la recherche opérationnelle, cette thèse fournit une revue complète, pour la première fois, sur l'utilisation des techniques d'AA dans la conception de différents éléments de MHs pour différents objectifs, y compris *sélection des algorithmes*, *évaluation de la fonction objective*, *initialisation*, *évolution* catégorisée en *sélection de l'opérateur*, *modèle d'évolution apprenable*, et *génération de voisins*, *paramétrage*, et enfin, *coopération*. De manière pédagogique, nous décrivons les concepts clés et les préliminaires de chaque mode d'intégration. De plus, nous passons en revue les avancées récentes sur chaque sujet et classons les articles de la littérature pour identifier les défis de la recherche. Nous proposons également une taxonomie pour fournir une terminologie et une classification communes. Une partie importante de l'étude consiste en une discussion technique sur les avantages et les limites, les exigences et les défis de la mise en œuvre de chaque mode d'intégration. Enfin, des directions de recherche prometteuses sont identifiées en fonction de la manière d'intégrer l'AA dans les MHs. Nous pensons que cette revue est indispensable non seulement pour les non-experts dans le domaine des MHs désirant utiliser des techniques d'AA, mais aussi pour les chercheurs seniors qui souhaitent enseigner aux étudiants juniors, en particulier les doctorants en recherche opérationnelle et en informatique.

AA-dans-MH : apprendre à sélectionner des opérateurs

Parmi les différents objectifs de l'intégration AA-dans-MH, nous nous concentrons sur *sélection d'opérateurs* dans cette thèse. La raison est le fait que l'utilisation des techniques d'AA pour la sélection des opérateurs de recherche a été un sujet de recherche actif ces dernières années et a montré des résultats prometteurs [San+14 ; MGS20]. Plus important encore, le succès de toute MH dépend fortement du choix/conception de ses opérateurs de recherche (c'est-à-dire la recherche locale et les opérateurs de perturbation). Par conséquent, dans cette thèse, nous nous concentrons sur la *sélection des opérateurs* et nous visons à proposer un cadre général à utiliser par n'importe quelle MH pour sélectionner le meilleur opérateur de recherche lorsqu'il s'agit d'opérateurs de recherche multiple utilisant des techniques d'AA.

Tout d'abord, discutons de la principale motivation pour laquelle les MHs peuvent avoir besoin d'utiliser plusieurs opérateurs. L'espace de recherche d'un POC est un environnement non stationnaire comprenant différentes régions de recherche aux caractéristiques dissemblables [Fia10]. En conséquence, le comportement d'un opérateur individuel peut changer en fonction de la région explorée, de sorte que les bons opérateurs peuvent devenir médiocres dans certaines régions, et vice-versa. Par conséquent, l'utilisation d'une combinaison de plusieurs opérateurs avec des caractéristiques différentes devrait améliorer la performance globale des MH, ce qui a été observé dans la résolution de différents POC [San+14 ; Li+15 ; DT+15]. En outre, l'utilisation d'une combinaison de divers opérateurs de recherche conduit à un processus de recherche plus étendu et permet de découvrir de nouvelles zones où de bonnes solutions peuvent être trouvées. Par conséquent, cela peut améliorer les capacités d'exploration et d'exploitation des MHs.

Lors de la conception de MHs avec de multiples opérateurs de recherche, une question majeure se pose : *dans quel ordre les opérateurs de recherche doivent-ils être employés pour guider efficacement la MH vers la solution optimale ?* Cette question est désignée

sous le nom de problème de *sélection des opérateurs*. Pour résoudre ce problème, deux mécanismes principaux peuvent être utilisés. Le mécanisme de sélection de l'opérateur peut être soit *hors ligne* (offline en anglais), soit *en ligne* (online en anglais). Dans une sélection d'opérateurs hors ligne, l'ordre des opérateurs de recherche est déterminé a priori et reste fixe tout au long du processus de recherche. Au contraire, dans la sélection d'opérateurs en ligne, les opérateurs sont sélectionnés et employés de manière dynamique pendant le processus de recherche. Dans la littérature, l'accent a été mis davantage sur la sélection d'opérateurs en ligne, ce qui est plus compatible avec l'hypothèse de non-stationnarité de l'espace de recherche.

L'approche de sélection d'opérateurs en ligne la plus simple consiste à sélectionner les opérateurs de manière aléatoire et uniforme à chaque étape de la recherche, sans tenir compte de leurs performances au cours du processus de recherche. Cependant, une telle sélection donne une chance égale à chaque opérateur d'être sélectionné, ignorant la bonne ou la mauvaise performance de l'opérateur. Contrairement à une telle sélection aveugle, une approche efficace consiste à prendre en compte les performances des opérateurs de recherche dans le processus de sélection. À cet égard, *Sélection Adaptative d'Opérateurs* (SAO) est une approche de sélection en ligne qui sélectionne et emploie dynamiquement les opérateurs les plus appropriés en fonction de l'historique de leurs performances pendant le processus de recherche [Fia10]. La SAO comporte deux tâches principales : le *affectation de crédit* qui attribue un crédit aux opérateurs en fonction de leur impact sur le processus de recherche et le *sélection d'opérateur* qui sélectionne le prochain opérateur à appliquer lors de l'itération suivante. Les méthodes pour réaliser la SAO diffèrent dans leur mécanisme d'affectation de crédit, des méthodes simples basées sur les scores [MJTM19 ; LT16 ; Zan+09], aux méthodes plus avancées utilisant des techniques d'AA [MGS20 ; Ahm+18 ; BEB17 ; San+14]. Comme étudié et montré par [TSH21], la SAO avec des méthodes d'affectation de crédits simples n'apportent pas de valeur ajoutée significative à la performance globale des MHs.

Pour surmonter l'inconvénient de la méthode d'affectation de crédits simple, les techniques d'AA telles que l'apprentissage par renforcement sont capables de trouver le meilleur comportement des MH (c'est-à-dire de sélectionner le meilleur opérateur) en utilisant l'expérience acquise en interagissant avec l'environnement. Une telle intégration AA-dans-MH pour la sélection de l'opérateur n'a pas été étudiée techniquement dans la littérature, et il y a un besoin d'études de recherche pour aborder ce problème en profondeur et étudier les défis, les limites et les opportunités à cet égard. Pour combler ce problème, nous proposons un cadre général dans lequel nous utilisons l'apprentissage par renforcement, plus précisément l'algorithme Q-learning, pour la SAO qui peut être employé dans n'importe quel MH. Ce cadre est applicable à un large éventail de POC sans nécessiter d'adaptations majeures. Nous évaluons les performances du cadre proposé en l'appliquant à la résolution de deux problèmes NP-hard, le problème du voyageur de commerce (Traveling Salesman Problem - TSP en anglais) et le problème d'ordonnancement de type flowshop de permutation (Permutation flow-shop scheduling problem - PFSP en anglais). En outre, les exigences, les défis et les limites de ce cadre sont discutés en détail. Cela aide les chercheurs à utiliser ce cadre pour leurs propres POC.

Cadre générale proposé

Nous proposons un cadre pour montrer comment l'AA peut être intégré dans les MHs pour sélectionner automatiquement les opérateurs de recherche sans injecter les connaissances des experts dans le processus de sélection. Parfois, les connaissances des experts ne sont pas suffisantes/optimales et peuvent conduire à des décisions dont les résultats ne

sont pas satisfaisants. Considérons un MH avec un ensemble d'opérateurs de recherche multiples. L'objectif de la sélection des opérateurs est de décider quel opérateur doit être sélectionné et appliqué à chaque étape de la recherche. La qualité (performance) des opérateurs à chaque étape de la recherche dépend de deux critères : premièrement, la nature stochastique du processus d'évolution, où certains opérateurs apparemment peu performants peuvent simplement avoir été malchanceux à certaines étapes, et deuxièmement, la région de l'espace de recherche explorée et ses caractéristiques, où un bon opérateur peut présenter une performance médiocre dans certaines régions. Par conséquent, il serait difficile de prévoir a priori la performance des opérateurs à chaque étape de la recherche et de déterminer le meilleur opérateur à l'aide de la connaissance experte, puisque cette performance dépend des caractéristiques de l'espace de solution de l'instance du problème à traiter ainsi que de la nature stochastique de l'évolution. Par conséquent, la connaissance experte ne pourrait pas nous donner l'opérateur optimal et donc la décision optimale.

Pour résoudre ce problème, nous proposons de remplacer les connaissances des experts par des techniques d'AA pour sélectionner automatiquement les opérateurs. Parmi les différentes techniques d'AA (apprentissage supervisé, apprentissage non supervisé, apprentissage semi-supervisé, et apprentissage par renforcement), nous proposons d'utiliser l'apprentissage par renforcement. Dans l'apprentissage supervisé, l'expert doit fournir l'ensemble des données d'entrée ainsi que les données de sortie attendues (comportement) pour entraÃőner le modèle d'AA et le but est d'imiter aveuglément l'expert et non d'optimiser une certaine mesure de performance. Cependant, il arrive que les connaissances de l'expert ne soient pas optimales/satisfaisantes. Dans ce cas, l'apprentissage par renforcement peut entrer en jeu pour former un modèle par essais et erreurs sans avoir besoin de connaissances spécialisées. Ce modèle est capable d'explorer l'espace des décisions possibles et d'apprendre par l'expérience le comportement le plus performant. En outre, dans l'apprentissage supervisé, le comportement appris peut ne pas être généralisé à des cas non vus. Au contraire, dans l'apprentissage par renforcement, pour une récompense donnée, le modèle apprend à trouver le meilleur comportement pour chaque instance.

La Figure 1 montre le mécanisme du cadre proposé. D'après la Figure 1, le cadre proposé comporte deux composantes principales : une technique d'AA et un algorithme méta-heuristique. Pour le composant d'AA, nous avons utilisé le Q-learning, une technique basée sur l'apprentissage par renforcement. En ce qui concerne la méta-heuristique, nous avons utilisé la recherche locale itérée (Iterated Local Search - ILS en anglais). ILS est une méta-heuristique simple basée sur une solution unique et puissante à la fois dans l'exploitation et l'exploration. Il convient de mentionner que le cadre proposé est général et qu'il peut être intégré à n'importe quelle méta-heuristique, qu'elle soit basée sur une solution unique ou sur une population. De même, il peut être appliqué pour résoudre tout problème d'optimisation combinatoire. Considérant le Q-learning comme la technique d'AA et l'ILS comme la MH, nous appelons le cadre proposé QILS.

Le cadre QILS proposé commence par la composante MH, où une solution initiale est d'abord générée. La solution passe ensuite par trois étapes principales de l'ILS : l'étape de perturbation (exploration), l'étape de recherche locale (exploitation) et l'acceptation. Une technique d'apprentissage par renforcement a été intégrée à l'ILS pour sélectionner l'opérateur le plus approprié parmi un ensemble de multiples opérateurs disponibles à appliquer à la solution. Par conséquent, selon l'objectif de l'utilisateur, l'apprentissage par renforcement peut être intégré à l'étape de perturbation, aux étapes de recherche locale, ou aux deux. Pour simplifier, dans la Figure 1, nous montrons un cas où l'algorithme de Q-learning, en tant que technique d'apprentissage par renforcement, a été utilisé pour sélectionner les opérateurs de perturbation dans l'étape de perturbation.

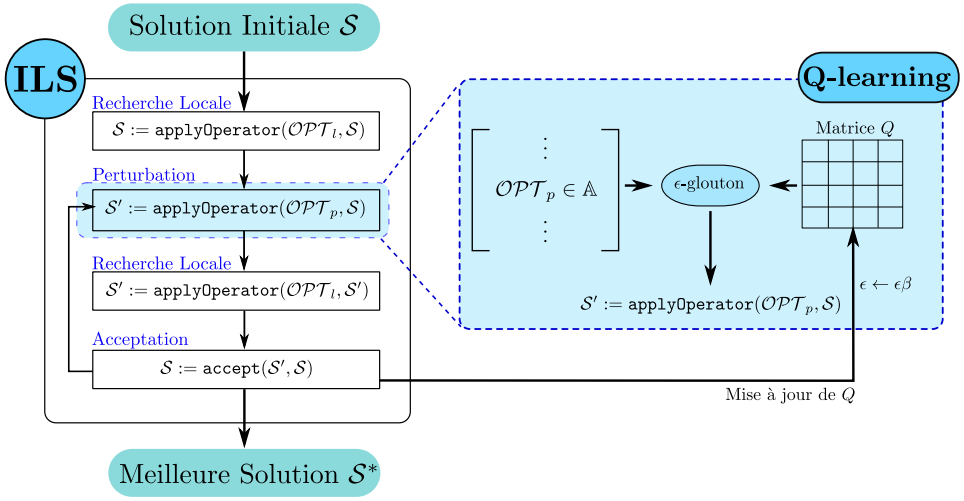

Figure 1 – Cadre QILS proposé

Dans notre processus de sélection, nous utilisons l'idée de la SAO, dans laquelle les opérateurs sont sélectionnés de manière adaptative tout au long du processus de recherche en fonction de leur qualité. En outre, nous prenons en compte le concept d'états dans l'apprentissage par renforcement dans notre processus de sélection. Les opérateurs sont sélectionnés en fonction de deux critères à chaque point de décision : l'historique des performances des opérateurs depuis le début de la recherche qui représente la qualité de l'opérateur, et l'état actuel (statut) de la recherche qui caractérise l'environnement étudié. Nous modélisons la sélection des opérateurs en utilisant l'algorithme Q-learning. Le Q-learning est composé de trois éléments principaux : l'ensemble des états $s \in \mathbb{S}$ qui représentent l'état de l'environnement, l'ensemble des actions $a \in \mathbb{A}$ qui représentent l'ensemble des décisions qui doivent être apprises par l'interaction avec l'environnement, et enfin, une fonction de récompense qui vise à guider le modèle pour prendre des décisions qui maximisent la récompense cumulative. Dans QILS, les actions (décisions) sont les opérateurs (perturbation, recherche locale) qui doivent être sélectionnés par Q-learning. D'autre part, l'ensemble des états et la fonction de récompense sont des propriétés spécifiques au problème qui sont déterminées en fonction du contexte et de l'objectif de l'utilisation du Q-learning.

Dans QILS, l'application de toute action est suivie d'une récompense immédiate de l'environnement (c'est-à-dire le problème) qui montre l'impact immédiat de l'opérateur appliqué sur le processus de recherche. Ensuite, sur la base de cette récompense immédiate et de l'historique des performances des opérateurs, l'algorithme Q-learning affecte un crédit à chaque opérateur. En fonction de l'état actuel de la recherche ainsi que du crédit de chaque opérateur, le prochain opérateur de perturbation est sélectionné pour être appliqué. Pour la sélection, nous utilisons la stratégie ϵ-glouton pour trouver un équilibre entre l'exploitation des opérateurs qui ont bien fonctionné jusqu'à présent et l'exploration d'autres opérateurs pour leur donner une chance d'être sélectionnés. Nous introduisons le concept d'épisode dans le cadre proposé, dans lequel chaque action a une chance d'un épisode égal à un nombre fixe d'itérations avant d'évaluer sa performance.

Analyse Empirique

Nous faisons des expériences complètes pour étudier les performances de l'algorithme QILS. Les expériences sont menées sur une grande variété d'instances de TSP et PFSP

afin de répondre à quatre questions de recherche : 1) Est-ce que l'incorporation de plusieurs opérateurs de recherche locale (perturbation) améliore la capacité d'exploitation (exploration) et les performances de l'ILS, et dans quelle mesure ? 2) Est-ce que l'utilisation de l'algorithme de Q-learning pour sélectionner les opérateurs appropriés améliore les performances de l'ILS, et dans quelle mesure ? 3) Dans quelle mesure l'algorithme de QILS proposé est-il compétitif par rapport aux algorithmes de la littérature ?, et 4) Comment l'algorithme QILS proposé adapte-t-il automatiquement les opérateurs à l'instance du problème à traiter ?

Pour le TSP, nous avons utilisé un ensemble d'instances de TSP symétriques sélectionnées au hasard dans la bibliothèque TSPLIB [Tsp] avec un nombre de villes différent allant de 50 à 2150 villes. Pour le PFSP, nous avons utilisé trois jeux de données bien connus, à savoir les jeux de données *Taillard* (contenant de 120 instances classées en 12 ensembles d'instances allant de 20 tâches avec 5 machines à 500 tâches avec 20 machines), *VRF-hard-small* (contenant de 240 instances réparties en 24 ensembles d'instances allant de 10 tâches et 5 machines à 60 tâches et 20 machines), et *VRF-hard-large* (contenant de 240 instances de grande taille classées en 24 ensembles d'instances allant de 100 travaux et 20 machines à 800 travaux et 60 machines).

Compte tenu de la nature stochastique des algorithmes et pour étudier la robustesse des algorithmes QILS pour trouver les solutions (quasi) optimales, chaque algorithme a été exécuté 30 fois indépendamment pour chaque instance et la moyenne des résultats a été calculée. Trois mesures clés de comparaison des performances sont utilisées pour comparer les performances des algorithmes : L'écart relatif moyen en pourcentage (Average Relative Percentage deviation - ARPD en anglais), le temps de calcul moyen, l'écart-type et le comportement de convergence.

Différentes comparaisons ont été effectuées pour la TSP et la PFSP. Pour la TSP, nous comparons l'algorithme QILS avec ses versions sans apprentissage (l'algorithme ILS avec des opérateurs de recherche locale (perturbation) uniques, et l'algorithme ILS avec le même ensemble d'opérateurs de recherche locale (perturbation) que QILS sélectionnés uniformément et aléatoirement). Pour le PFSP, nous comparons le QILS avec ses versions sans apprentissage, ainsi que sept algorithmes de la littérature.

Conclusion et discussion

Comme première contribution, des défis particuliers et des directions de recherche futures ont été élaborés pour chaque mode d'intégration AA-dans-MH tout au long du manuscrit. En outre, il existe un ensemble de défis communs et de perspectives futures, quelle que soit la méthode d'intégration.

Chaque fois qu'une technique d'AA est intégrée dans un MH, une série de paramètres supplémentaires sont introduits et doivent être soigneusement réglés/contrÃŧlés pour obtenir les meilleures performances. La définition de la valeur de ces paramètres supplémentaires peut augmenter l'effort de calcul de l'étape de définition des paramètres, cependant, la définition des paramètres est généralement une tâche unique et le gain que les techniques d'AA apportent en économisant l'effort de calcul du processus de recherche peut compenser l'effort de calcul supplémentaire de la définition des paramètres.

Plus le volume de données est important, plus les performances des techniques d'AA sont élevées. La disponibilité des données est un autre défi lors de l'intégration des techniques d'AA dans les MHs. En effet, la collecte ou même la génération de suffisamment de données est une tâche difficile. Même si suffisamment de données historiques sont disponibles, la façon d'échantillonner à partir des données historiques pour imiter de manière

appropriée le comportement reflété dans ces données est un autre défi [BLP21a]. Une façon de relever le défi de la disponibilité des données pourrait consister à utiliser *Few-Shot Learning* pour former un modèle avec une très petite quantité de données de formation [Wan+20a]. En utilisant la connaissance préalable d'instances de problèmes similaires, l'apprentissage à quelques instants peut être rapidement généralisé à de nouvelles tâches ne contenant que quelques instances de problèmes avec des informations supervisées.

La majorité des études dans la littérature utilisent des techniques classiques d'AA telles que la méthode des k plus proches voisins (k-Nearest Neighbours en anglais), k-means, les machines à vecteurs de support (support vector machines en anglais), la régression logistique (logistics regression en anglais), etc. Avec le développement rapide des nouvelles technologies, les problèmes du monde réel deviennent de plus en plus complexes, et avec les nouvelles avancées de la numérisation, diverses données en temps réel sont collectées massivement qui ne peuvent pas être traitées par les techniques d'AA classiques. Ces données volumineuses soulèvent plusieurs problèmes qui doivent être pris en considération [Emr16]. Pour faire face à de telles données volumineuses, des techniques d'AA plus avancées telles que l'apprentissage profond peuvent être intégrées dans les MH. À cet égard, lorsque diverses techniques d'AA sont disponibles pour être intégrées dans les MHs pour un but particulier, le problème de sélection d'algorithme peut être étudié pour sélectionner la technique d'AA la plus appropriée. En outre, avec le développement des superordinateurs, l'exploration du concept de parallélisme dans l'intégration des techniques d'AA dans les MHs à l'aide d'accélérateurs GPU (Graphics Processing Units en anglais) et TPU (Tensor Processing Units en anglais) pourrait être une direction de recherche future intéressante : [CMT04 ; Alb05 ; VLMT11].

En ce qui concerne notre deuxième contribution, nous avons montré que le cadre proposé obtient des résultats prometteurs à la fois pour le TSP et le PFSP. En ce qui concerne le TSP, nous avons observé que l'intégration de l'algorithme Q-learning améliore significativement les performances de l'ILS par rapport à ses versions sans apprentissage. En ce qui concerne le PFSP, l'algorithme QILS proposé obtient les meilleures performances. Nous tirons quelques conclusions et observations principales que nous avons obtenues par l'application à ces deux problèmes.

La principale conclusion à tirer de la comparaison des performances de QILS par rapport à sa version à sélection aléatoire est que l'utilisation de l'algorithme de Q-learning pour sélectionner automatiquement les opérateurs permet une amélioration significative des performances. En d'autres termes, la connaissance recueillie par le processus d'apprentissage tout au long du processus de recherche est précieuse et conduit à l'identification de l'opérateur le plus approprié à chaque point de décision et par conséquent au succès du cadre en convergeant plus rapidement vers de meilleures solutions. Par ailleurs, nous avons également observé que plus la taille des instances est élevée, plus la performance de notre algorithme est élevée.

Nous avons également observé que le QILS est presque insensible (ou pas très sensible) à la taille de l'ensemble des actions. Le fait que QILS ne soit pas très sensible à la taille des actions devient de plus en plus important et utile lorsqu'il n'y a pas de connaissances préalables suffisantes concernant les performances des opérateurs individuels. Par conséquent, QILS peut être exécuté avec un ensemble d'opérateurs différents, mais pas nécessairement les meilleurs ; le mécanisme de sélection proposé, basé sur l'algorithme de Q-learning, est alors capable de sélectionner automatiquement les opérateurs les plus appropriés, c'est-à-dire les actions, pendant le processus de recherche parmi tous les opérateurs disponibles.

Nous avons montré que l'ajout de l'apprentissage par renforcement n'augmente pas de

manière significative la complexité du cadre par rapport à la version de base de MH. En considérant la complexité dans le pire des cas, l'apprentissage par renforcement a un surcoÃżt de complexité de la taille de l'ensemble d'actions ($\mathcal{O}(|A|)$ qui est constant, et il ne dépend pas de la taille du problème. Même en pratique, nous avons montré que le cadre proposé obtient de meilleurs résultats avec un effort de calcul encore plus faible, en particulier pour les grandes instances, en raison d'une convergence plus rapide vers de bonnes solutions (quasi-optimales/optimales). Cette conclusion peut être généralisée à n'importe quel MH utilisé dans le cadre, et également appliquée à n'importe quel POC.

Un défi important dans l'algorithme de Q-learning est de définir un ensemble d'états en fonction de l'objectif de l'utilisation de Q-learning. Les états doivent être complètement descriptifs de l'état du problème pour permettre de sélectionner l'action correcte. Il y a trois façons de définir les états. Les états peuvent être 1) dépendants de la recherche qui reflètent les propriétés du processus de recherche, comme le nombre d'itérations non améliorées, 2) dépendants du problème qui reflètent les propriétés du problème par le biais de caractéristiques génériques, ou 3) dépendants de l'instance qui reflètent les propriétés de l'instance du problème, comme le nombre de bacs dans un problème d'emballage de bacs.

Afin de surmonter le défi de déterminer les opérateurs a priori pour l'algorithme de Q-learning, une direction de recherche future prometteuse est de considérer un ensemble dynamique d'opérateurs candidats, au lieu d'un ensemble statique avec des opérateurs fixes, pour gérer un grand ensemble d'opérateurs d'une manière en ligne. L'idée est que les opérateurs jugés inefficaces à certaines étapes de la recherche sont momentanément désactivés en quittant l'ensemble de candidats pour un nombre fixe d'itérations, et d'autres nouveaux opérateurs peuvent être invités à être inclus dans l'ensemble de candidats pour des étapes particulières de la recherche. Ce mécanisme permet d'inclure des opérateurs efficaces et d'exclure des opérateurs inefficaces à différents stades de la recherche afin d'obtenir les meilleures performances sans imposer de surcharge de calcul supplémentaire. Ce mécanisme est appelé gestion adaptative des opérateurs.

Abstract

In recent years, there has been a growing research interest in integrating machine learning techniques into meta-heuristics for solving combinatorial optimization problems. This integration aims to guide the meta-heuristics toward making better decisions and consequently make meta-heuristics more efficient and improve their performance in terms of solution quality, convergence rate, and robustness. The contribution of this thesis to this interdisciplinary research domain between Operations Research and Computer Science is an *analytical-technical* contribution.

From an *analytical* viewpoint, we provide, as the first of its kind, a comprehensive yet technical review on the research works addressing this integration. Through this review, we propose a unified taxonomy for different ways of integration, including *algorithm selection*, *fitness evaluation*, *initialization*, *evolution* categorized into *operator selection*, *learnable evolution model*, and *neighbor generation*, *parameter setting*, and *cooperation*. After a detailed classification of the papers corresponding to each class of integration, more importantly, a complete analysis and discussion is provided on technical details (i.e., challenges, advantages, disadvantages, perspectives, etc.) of each way of integration.

From a *technical* aspect, we focus on a particular integration and address the problem of *adaptive operator selection* in meta-heuristics using reinforcement learning techniques. More precisely, we propose a general framework that integrates the Q-learning algorithm, as a reinforcement learning algorithm, into the iterated local search algorithm to adaptively and dynamically select the most appropriate search operators at each step of the search process based on their history of performance. The proposed framework is general and can be applied to any meta-heuristics, where a set of competitive search operators exist for solving the problem.

To investigate the performance of the proposed framework, we applied it on two combinatorial optimization problems, traveling salesman problem and permutation flowshop scheduling problem. In both applications, the framework yields significant improvement in terms of solution quality and convergence rate compared to its non-learning version, where operators are selected randomly. Besides, we show that our algorithm performs even better when dealing with large size instances of the problems. Moreover, we observe that the proposed framework shows the state-of-the-art behavior when solving the permutation flowshop scheduling problem.

Chapter 1

Introduction

Contents

1.1 Introduction **28**
1.2 Contribution of thesis **30**
1.2.1 ML-into-MH: A taxonomy 30
1.2.2 ML-into-MH: Learning to select operators 31
1.3 Thesis outline **32**

1.1 Introduction

Combinatorial Optimization Problem (COP) is a class of optimization problems that is challenging to solve. The word Combinatorics is concerned with the selection, arrangement, or sequencing of a collection of objects. Combinatorial optimization is the search for the best selection, arrangement, or sequence with respect to some objective function in a finite search space. There are a variety of types of COPs that have been widely studied by the Operations Research community. One classical COP that has been widely studied is the Traveling Salesman Problem (TSP). In TSP, there is a salesman who performs a tour by visiting a set of n cities exactly once and returning to the starting point (i.e., Hamiltonian tour). Each edge connecting two cities i and j is associated with a distance d_{ij}, and the goal is to find the Hamiltonian tour of cities with the minimum total distance. Let's consider an exhaustive search (enumeration) algorithm that evaluates all possible Hamiltonian tours of a given TSP instance and returns the tour with the minimum total distance as the optimal solution. When the number of cities is low (e.g., $n \leq 6$), an exhaustive search algorithms is efficient and finds the optimal solution rapidly. However, as the number of cities increases, which is the case in real-world problems, it would become more and more time-consuming (if not computationally impossible) to employ an exhaustive search algorithm to find the optimal solution for TSP. To be more precise, the number of possible Hamiltonian tours for a TSP of n cities with a symmetric distance matrix between the cities is calculated as $(n-1)!/2$. Table 1 shows how the number of Hamiltonian tours increases by increasing the number of cities. According to Table 1, as the number of cities increases by even small values, the number of Hamiltonian tours increases explosively by an order of $(n-1)!/2$. It implies that the computational complexity of an exhaustive search algorithm is $\mathcal{O}(n!)$ for solving a TSP instance of size n. For example, for a small-sized TSP instance with $n = 20$ number of cities, an exhaustive search algorithm should evaluate $6.08006E+16$ different Hamiltonian tours. Considering that each evaluation takes only a μs, a TSP instance with $n = 20$ cities takes around twenty thousand centuries to be solved exhaustively.

Table 1 – Number of Hamiltonian tours for TSP

# of cities (n)	# of Hamiltonian tours
5	12
6	60
7	360
8	2520
9	20160
10	181440
20	6.08226E+16
25	3.10224E+23

TSP belongs to the NP-hard class of optimization problems for which no known resolution algorithm with a polynomial time complexity of $O(n^k)$ (for some k) exists [Woe03]. Similar to TSP, most COPs belong to the NP-hard class of optimization problems. Any NP-hard problem can be solved using exhaustive search. However, when the size of the instances grows, the running time for an exhaustive search becomes severely huge, even for instances of fairly small size in real-world.

Exhaustive search algorithms belong to a general class of algorithms called *Exact* algorithms [LN87; Woe03]. Exact algorithms (e.g., branch-and-bound, benders decom-

position, etc.), as their name implies, are algorithms that always solve an optimization problem to optimality [FK13]. For some large-sized TSP instances, there may exist exact algorithms able to solve them to optimality in a polynomial time. However, TSP and most COPs are NP-hard in general and solving real-world large-sized instances to optimality would be very hard and time-consuming (if not computationally impossible).

The inherent complexity of COPs enforces the need for approximate algorithms. Approximate algorithms are a class of optimization algorithms that are mainly developed to solve complex optimization problem (i.e., COPs) [GK06]. These algorithms are able to find (near-) optimal solutions in a reasonable computational time; however, they do not guarantee the optimality of the obtained solutions. For more details on the approximate algorithms, interested readers are referred to Section 2.2 in Chapter 2. Among different categories of approximate algorithms, Meta-heuristic (MH) algorithms are computational intelligence paradigms applicable to a large variety of optimization problems, if well-designed and tailored [OL96]. Considering their ability in finding (near-) optimal solutions for COPs in a reasonable computational time, they are good substitutes for exact algorithms [HW90; OL96; Tal09]. From a technical point of view, MHs are optimization algorithms that arrange and pilot an interaction between local improvement procedures and higher-level strategies to create an iterative search process capable to escape from local optima and perform a robust search of a search space [GP10]. During such an iterative search process, a considerable number of solutions are generated, evaluated, and evolved until a promising solution is obtained. In fact, during the search process, MHs generate a considerable volume of data including good (elite) or bad solutions in terms of their fitness values, the sequence of search operators from beginning to the end, evolution trajectories of different solutions, local optima, etc. These data potentially carry useful knowledge such as the properties of good and bad solutions, the performance of different operators in different stages of the search process, precedence of search operators, etc. Classical MHs do not use any form of knowledge hidden in these data to solve COPs. However, in the last decade, there has been a huge interest to make benefit of these data and extract useful knowledge from them to solve COPs. In this regard, numerous studies have investigated the use of Machine Learning (ML) techniques when solving COPs.

ML is a subfield of artificial intelligence whose primary concern is the design and analysis of algorithms which enable computers to learn [WBK20]. As a subfield of artificial intelligence, ML has the ability to learn and adapt to changes in a changing environment and accordingly to generalize well to new settings without the need to foresee and provide solutions for all possible settings. ML techniques autonomously learn how to perform a task or make predictions by detecting certain patterns or regularities using the training data or past expegrience, assuming that the future will not be much different from the past. ML can be used to teach computers to perform a wide variety of tasks, ranging from automatic object detection and speech recognition to robotics and predictive analytics used for forecasting purposes.

ML techniques can be used for solving COPs either directly as a solver or in an integrated way into exact and approximate solution algorithms. ML techniques can directly solve COPs either using their predictive ability to predict good/optimal solution for COPs [Jos+22; Ben+20] or distinguish between optimal and non-optimal solutions [AS19] or through the automatic design of good/fast algorithms from scratch able to solve different COP instances with different settings by exploiting common patterns in different problem instances [Kha+17]. On the other hand, ML techniques can be integrated into solution algorithms to extract useful knowledge from the generated data and inject it into the search process with the aim to improve the performance of the algorithms [BLP21b;

Kar].

When integrating ML techniques into MHs, the main goal is to improve the efficiency of MHs either through speeding up the search process by doing approximations or injecting knowledge into the search process or through finding better solutions by replacing the expert knowledge and experience with better decisions obtained using ML techniques. In this regard, various studies have been done on the integration of ML techniques into MHs for different purposes. Some studies focused on using classical ML techniques for conventional purposes, such as initial solution generation or approximating the fitness function. On the other hand, there are studies using more advanced ML techniques for automatic design of advanced MHs through cooperative frameworks. This integration can guide the MHs toward making better decisions and consequently make MHs more efficient and improve their performance in terms of solution quality, convergence rate, and robustness [Kar].

1.2 Contribution of thesis

In recent decade, the integration of ML techniques into optimization methods, particularly MHs, has gained a considerable interest which has yielded in numerous research articles. However, there are still plenty of research directions which can be further investigated [Tal02; STÃ19; BLP21b]. The contribution of this thesis to the domain of integrating ML techniques into MHs (ML-into-MH) lays in two *analytical* and *technical* levels. As an *analytical* level, this thesis provides as the first of its kind a comprehensive review on the ML-into-MH integration that not only classifies and reviews different research articles with different integration purposes but also provides a complete analysis and discussion on technical details of this integration (i.e., challenges, advantages, disadvantages, opportunities, etc.). In the *technical* level, we focus on a particular integration purpose, where we investigate the integration of ML techniques to guide MH through selecting the most appropriate search operator at each step of the search process. These contributions are discussed in more details in the two following subsections.

1.2.1 ML-into-MH: A taxonomy

Due to an increasing interest in the integration of ML techniques into MHs, numerous research articles have been presented in this domain, especially in the recent decade. Accordingly, different researchers have provided review studies to study the recent advances in this domain [JDT06; Zha+11; CDJ12; Tal16; Cal+17; STÃ19; Tal20]. Although there are different review studies in the literature, we believe that there is still a lack of a review study to go beyond mere description of the literature, to take a critical view to investigate the technical aspects of such integration by focusing on a set of challenges, guidelines, and limitations that researchers may face, and also to foster future research by developing new ideas and insights.

In order to fill this gap, we provided such a review with the mentioned characteristics. Our proposed review study contributes to the literature from three aspects: first, it serves as a pedagogical study that is a good starting point for the researchers who intend to begin doing research in this field, and second, it is a technical study that identifies and clarifies, without any bias, the technical challenges, requirements, and limitations of this integration, and finally, by providing a broader view over this integration, it proposes new ideas and research insights to deal with the identified challenges. The main features of our work that distinguish it from the available review studies in the literature have been elaborated in detail in Section 3.1.1 in Chapter 3.

According to our proposed taxonomy, ML techniques can be integrated into MHs for different purposes including *algorithm selection*, *fitness evaluation*, *initialization*, *evolution* categorized into *operator selection*, *learnable evolution model*, and *neighbor generation*, *parameter setting*, and finally, *cooperation*. Each category has been explained in detail in Chapter 3.

1.2.2 ML-into-MH: Learning to select operators

Importantly, the success of any MH highly depends on the choice/design of its search operators (i.e., local search and perturbation operators), and MHs with multiple operators have shown better performance when solving COPs [San+14; Li+15; DT+15]. In fact, the search space of a COP is a non-stationary environment including different search regions with dissimilar characteristics [Fia10]. Accordingly, the behavior of an individual operator may change depending on the region being explored such that good operators might become poor at some regions, and vice-versa. Hence, it is reasonable to expect that employing different operators combined appropriately during the search process will produce better solutions. Accordingly, among different purposes of the ML-into-MH integration, we focus on *operator selection* in this thesis, with the aim to help MHs to select the most appropriate search operator at each step of their search process. Moreover, the use of ML techniques for selecting the search operators in MHs has been an active research topic in recent years and has shown promising results [San+14; MGS20]. Therefore, in this thesis, we focus on *operator selection* and aim to propose a general framework to be used by any MH to automatically select the most appropriate search operator, when dealing with multiple search operators, using ML techniques.

In such framework, a major question arises: *in which order should the search operators be employed to efficiently guide the MH toward the optimal solution?* The order of operators can be determined either a priori (i.e., *offline* selection) or during the search process (i.e., *online* selection). In the literature, there has been more focus on online operator selection, which is more compatible with the non-stationarity of the search space of COPs. Through an online selection, the operators can be selected either randomly (i.e., blind selection) or dynamically based on the performance of the operators. The latter is referred as to Adaptive Operator Selection (AOS). In an AOS, an important task is to assign a credit to each operator based on their performance. This assignment plays an important role in the success of an AOS and can be done using either simple score-based methods [MJTM19; LT16; Zan+09], to more advanced methods using ML techniques [MGS20; Ahm+18; BEB17; San+14]. [TSH21] has done a meta-analysis on the added-value of using simple credit assignment methods, with focus on the adaptive large neighborhood search MH. According to their analysis, AOS with simple credit assignment methods do not bring significant added-value to the overall performance of the MH.

To overcome the drawback of the simple credit-assignment methods, ML techniques such as Reinforcement Learning (RL) can be employed. RL is a ML technique that sequentially learns, through trial-and-error, the best action to take at a given state to achieve a goal. In RL, there is an agent that interacts with an environment and iteratively learns the best action to take to maximize the cumulative reward (i.e., feedback received from the environment as a result of performing an action) through this interaction. Using RL for AOS, the agent would be able to learn the best operator (i.e., action) to select at each decision point based on the experience gained through interaction with the environment. Such ML-into-MH integration for operator selection has not been technically studied in the literature, and there is a need for research studies to address this problem and study the challenges, limitations, and opportunities in this regard. To

fill this gap, we propose a general framework where we use RL, more specifically the Q-learning algorithm, for AOS that can be employed in any MHs. This framework is applicable to a wide range of COPs without the need for any major adaptations. We assess the performance of the proposed framework by applying it to solve two NP-hard COPs, TSP and Permutation Flowshop Scheduling Problem (PFSP). Furthermore, the requirements, challenges, and limitations of this framework are thoroughly discussed, and future research directions have been proposed. This technical discussion helps the researchers to employ the proposed framework to solve their own COPs.

1.3 Thesis outline

In this chapter, we introduced the subject of the thesis and elaborated two main contributions of this thesis to the literature. To realize these contributions, the following outline is proposed.

First, Chapter 2 "*Theoretical Background*" introduces the main concepts and methods used in this thesis including MHs, ML techniques and their different types, as well as the COPs. These are the three main elements of this thesis. In addition to the general definitions provided in Chapter 2, we explain in detail the formal definition and formulation of two COPs (i.e., TSP and the PFSP), as well as the Iterated Local Search (ILS) as the MH and Q-learning as the ML technique that we used in this thesis.

After defining and explaining the prerequisites, Chapter 3 "*Integration of Machine Learning into Meta-heuristics: A Taxonomy*" realizes the first contribution of this thesis and provides a taxonomy and classification of different purposes for which ML techniques can be integrated into MHs. Each purpose is then explored in detail, and valuable technical discussion and insights are provided for each different purpose.

After providing a comprehensive and technical overview on the ML-into-MH integration, the readers will follow Chapter 4 "*Q-learning for Operator Selection: A General Framework*" that develops a general framework for the purpose of operator selection using ML techniques. The framework is first designed and characterized, then applied to solve two NP-hard COPs, TSP and PFSP. Through the application to these two problems, we analyze the gains of integrating ML techniques into MHs for AOS. We finally show that the proposed operator selection framework outperforms its non-learning variants in terms of solution quality and convergence rate, and shows the state-of-the-art behavior for PFSP.

Finally, Chapter 5 "*Conclusions and Future Research*" concludes this thesis, discusses the obtained results, and proposes promising research directions, especially on the generalization of the proposed framework to other COPs. This generalization has been addressed partially through a technical work that we have done, which will be briefly discussed at the end.

Chapter 2

Theoretical Background

Contents

2.1 Combinatorial optimization problems 34
2.1.1 Traveling salesman problem 34
2.1.2 Permutation flowshop scheduling problem 35
2.2 Meta-heuristics 36
2.3 Machine learning 39
2.3.1 Supervised learning algorithms 39
2.3.2 Unsupervised learning algorithms 40
2.3.3 Semi-supervised learning 41
2.3.4 Reinforcement learning algorithms 41
2.4 Adaptive Operator Selection & Learning 43

In this chapter, we introduce the main concepts and methods used in this thesis, including the COPS, MHs, and ML techniques. We start, first, by providing a formal definition of COPs and their inherent complexity, and present the mathematical formulations of two COPs we used as our application, TSP and PFSP. Next, we continue by defining approximate algorithms in general, their main categories and the specific features of each category. We focus on MHs, as a specific category of approximate algorithms, and define the common concepts of MHs. We also explain in detail the procedure of ILS as the MH that we used in this thesis. Then, we define ML techniques and explain different types of ML techniques including supervised, unsupervised, semi-supervised, and RL techniques. Considering that we have used the Q-learning as our ML technique in this thesis, we elaborate on the Q-learning, the concepts of states, action, and the reward function. Finally, we provide a general overview on the operator selection paradigm, AOS, and the role of ML techniques in AOS.

2.1 Combinatorial optimization problems

In general, optimization problems can be seen as decision problems, where the solutions are additionally evaluated by a function, called the objective function. The goal is then to find a solution, among an infinite number of feasible solutions, with optimal objective function value (i.e., the optimal solution).

Combinatorial optimization is an emerging field at the cutting edge of combinatorics and theoretical computer science that attempts to use combinatorial techniques to solve discrete optimization problems. Accordingly, any combinatorial optimization problem (COP) can be stated as an optimization problem with discrete decision variables, where the number of feasible solutions is finite, although still too large for an exhaustive search to be a realistic option [Kor+12]. The formal representation of a COP is as follows:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } x \in S \end{aligned} \tag{2.1}$$

where $\mathcal{S}$ is the finite set of feasible solutions to the problem such that $\mathcal{S} \subset \mathbb{Z}^n$, $n \in \mathbb{Z}^+$, and $f : X \to \mathbb{Z}$. $\mathbb{Z}$ is the set of integers, $\mathbb{Z}^+$ is the set of positive integers, and $\mathbb{Z}^n = \mathbb{Z} \times \mathbb{Z} \times ... \times \mathbb{Z}$ is the set of n-vectors with integer components.

Most COPs belong to the NP-hard class of optimization problems, which require exponential time to be solved to optimality [Tal09]. Our proposed operator selection framework has been applied to solve two NP-hard COPs, TSP and PFSP whose definition and mathematical formulation are presented in the following.

2.1.1 Traveling salesman problem

TSP is a classical NP-hard COP that was first introduced by an Irish mathematician Sir William Rowan Hamilton and a British mathematician Thomas Penyngton Kirkman [BLW86]. The general form of TSP has been first studied by the mathematicians, mainly Karl Menger during the 1930s. TSP has a wide variety of applications in real-world. TSP naturally arises in transportation and logistics, allowing to address the problems arising in everyday life for the ordinary people (the problem of arranging school bus routes) as well as the transportation companies providing delivery services to the customers (the problem of parcel delivery). This is the main reason why TSP has been widely studied by many researchers from the Computer Science and Operations Research communities.

TSP can be formally defined by means of a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, 2, ..., v\}$ is the set of vertices representing cities and $\mathcal{E} = \{\langle i, j\rangle : i, j \in \mathcal{V}\}$ is the set of edges that connect the vertices of $\mathcal{V}$. The edge that connects cities i and j has a weight of $d_{ij} \in D$, which represents the distance between cities i and j; $i, j \in \mathcal{V}$. A solution to the TSP is a tour of cities $\Gamma := (\gamma_1, \gamma_2, \ldots, \gamma_v) \in \mathfrak{S}(\mathcal{V})$, where $\gamma_j \in \mathcal{V}$ denotes the index of the city appearing at position j in the tour Γ. In TSP, the aim is to find a Hamiltonian tour Γ of the minimum total travel distance z such that all vertices are visited exactly once. The mathematical formulation of TSP has given in Equations (2.2)-(2.6).

$$\min z = \sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{V}|} d_{ij} x_{ij} \tag{2.2}$$

$$\sum_{i=1}^{|\mathcal{V}|} x_{ij} = 1 \qquad \forall j \in \mathcal{V} \tag{2.3}$$

$$\sum_{j=1}^{|V|} x_{ij} = 1 \qquad \forall i \in \mathcal{V} \tag{2.4}$$

$$\sum_{i,j \in \mathcal{H}} x_{ij} \leq |\mathcal{H}| - 1 \qquad \forall \mathcal{H} \subset \mathcal{V} \tag{2.5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \neq j \in \mathcal{V} \tag{2.6}$$

Equation (2.2) defines the total travel distance z to be minimized. Equations (2.3) and (2.4) ensure that each city is visited exactly once. Accordingly, each city has a unique precedent and subsequent cities. Equation (2.5) ensures that a tour is fully connected by eliminating the sub-tours and finally, Equation (2.6) is the binary integrality constraint.

2.1.2 Permutation flowshop scheduling problem

The flowshop scheduling problem is a well-known class of scheduling problems, where a set $\mathbb{N} = \{1, 2, \ldots, n\}$ of n independent jobs have to be processed on a set $\mathbb{M} = \{1, 2, \ldots, m\}$ of m machines. When the sequence of jobs is the same for all machines, the problem is denoted as PFSP. PFSP is a conventional optimization problem that arises in most production systems. A solution to the PFSP is a permutation $\Pi := (\pi_1, \ldots, \pi_n) \in \mathfrak{S}(\mathbb{N})$ of the jobs, where $\pi_j \in \mathbb{N}$ denotes the index of the job appearing at position j in the sequence Π. Let $p_{i\pi_j}$ denote the processing time of job π_j on machine i. The completion time of job π_j on machine i, $C_{i\pi_j}$, can be determined in a recursive way using Equations (2.7)-(2.10):

$$C_{1\pi_1} = p_{1,\pi_1}, \tag{2.7}$$

$$C_{1\pi_j} = C_{1,\pi_j - 1} + p_{1,\pi_j}, \qquad \forall j \geq 2 \tag{2.8}$$

$$C_{i\pi_1} = C_{i-1,\pi_1} + p_{i,\pi_1}, \qquad \forall i \geq 2 \tag{2.9}$$

$$C_{i\pi_j} = \max\{C_{i-1,\pi_j}, C_{i,\pi_j - 1}\} + p_{i,\pi_j}, \qquad \forall i \geq 2, j \geq 2 \tag{2.10}$$

Solving a PFSP involves determining the permutation Π such that a particular objective is optimized [Joh54]. Different objectives can be considered for the PFSP such as the

minimization of *makespan*, *total flow time*, *total tardiness* or *sum of the total earliness and tardiness* [FVMPF20]. In this thesis, we focus on the makespan objective function C_{max}, given in Equation (2.11), which is one of the most complex variants among the variants with different objective functions and has proven to be NP-hard even for very small-sized instances ($m > 2$) [RS07]. Later in this work, we use $C_{max}(\Pi)$ as a convenient notation to denote the makespan of a solution Π.

$$C_{max} = C_{m\pi_n} \tag{2.11}$$

2.2 Meta-heuristics

As elaborated in Chapter 1, solving a large number of real-life COPs (e.g., TSP) in an exact manner is intractable within a reasonable amount of computational time. Approximate algorithms are then alternatives to solve these problems. Although approximate algorithms do not guarantee the optimality, their goal is to obtain solutions as close as possible to the optimal solution in a reasonable amount of computational time, at most polynomial [Tal09].

Approximate algorithms are divided into two classes of *approximation* algorithms and *heuristic* algorithms [Tal09]. The *approximation* algorithms are efficient algorithms that employ approximate rules and find approximate solutions to optimization problems with provable guarantees on the distance of the obtained solution to the optimal one [WS11]. For instance, the class of k-approximation algorithms for solving the TSP, return a tour whose cost is never more than k times of the cost of an optimal tour. The 2-approximation and 3/2-approximation algorithm are algorithms to approximate the solution of the TSP. The former approximates a Hamiltonian tour based on the idea of minimum spanning tree that is not twice worse than the optimal solution. The complexity of the 2-approximation algorithm is $O(n^6)$ when using Lawler's matroid intersection algorithm [Law01]. With a complexity of $O(n^3)$, the 3/2-approximation algorithm yields a tour that is at most 3/2 times longer than the optimal one [Chr76].

Unlike the approximation algorithms, for *heuristic* algorithms, there is no known efficient way to find an approximate solution with a given upper-bound on its gap to optimality. These heuristic algorithms are either problem-specific heuristics or MHs. Problem-specific heuristics are designed for and are applicable to a particular problem at hand. For example, the Lin-Kernighan Heuristic [LK73] is a problem-specific heuristic algorithm for TSP. On the other hand, MHs represent more general algorithms applicable to a large variety of optimization problems, if well-designed and tailored. Generally, MHs need to be provided with expert knowledge and tailored to a specific problem domain.

The term meta-heuristic was first introduced by Glover in 1986 [Glo86]. The word heuristic (from an old Greek work heuriskein or euriskein, to search) signifies "the art of discovering new strategies (rules) to solve problems", and the word meta (from the Greek prefix meta, beyond in the sense of high-level) means "upper level methodology". Therefore, the word meta-heuristic is defined as an "upper level general methodology that can be used as a guiding strategy in designing underlying heuristics to solve specific optimization problems" [SG13]. More technically, MHs are "solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space" [GP10].

MHs can be classified in different ways. They can be *nature-inspired* or *non-nature inspired* [Tal09]. Many MHs are inspired by natural phenomena. Evolutionary Algorithms (EAs) such as Genetic Algorithm (GA) [Hol+92], Memetic Algorithm (MA) [Mos+89], and Differential Evolution (DE) [SP97] are inspired by biology; Artificial Bee Colony (ABC) [Kar05], Ant Colony Optimization (ACO) [DB05], and Particle Swarm Optimization (PSO) [Ken06] are inspired by swarm intelligence; and Simulated Annealing (SA) [KGV83] from physics. There are also MHs inspired by non-natural phenomena; Imperialist Competitive Algorithm (ICA) [AGL07] from society, and Harmony Search (HS) [GKL01] from musics. From another perspective, MHs can be *memoryless*, or they may use *memory* during the search process [Tal09]. Memoryless MHs (e.g., GA, SA, etc.) do not use the historical information dynamically during the search process. However, MHs with memory, such as Tabu Search (TS) [GL98], memorize historical information during the search process, and this memory helps to avoid making repetitive decisions.

Furthermore, MHs can make *deterministic* or *stochastic* decisions during the search process to solve optimization problems [Tal09]. MHs with deterministic rules (e.g., TS) always obtain the same final solution when starting from the same initial solution, while stochastic MHs (e.g., SA, GA) apply random rules to solve the problem and obtain different final solutions when starting from the same initial solution. Moreover, in terms of their starting point, MHs are divided into *single-solution based* or *population-based* MHs [Tal09]. Single-solution based MHs, also known as trajectory methods, such as ILS [LMS03], Breakout Local Search (BLS) [BEB17], Descent-based Local Search (DLS) [ZHD16], Guided Local Search (GLS) [VT99], Variable Neighborhood Search (VNS) [MH97], Hill Climbing (HC) [JPY88], Large Neighborhood Search (LNS) [Sha98], Great Deluge (GD) [Due93], TS, SA, etc., manipulate and transform a single solution to reach the (near-) optimal solution. Population-based MHs such as Water Wave Optimization (WWO) [Zhe15], GA, PSO, ACO, etc., try to find the optimal solution by evolving a population of solutions. Because of this nature, population-based MHs are more exploration search algorithms, and they allow a better diversification in the entire search space. Single-solution based MHs are more exploitation search algorithms, and they have the power to intensify the search in local regions.

Finally, depending on their search mechanism, MHs can be *iterative* or *greedy* [Tal09]. The former (e.g., ILS, GA) starts with a complete solution and manipulates it at each iteration using a set of search operators. The latter, also called constructive algorithm, starts from an empty solution and constructs the solution step by step until a complete solution is obtained. Classical examples of greedy algorithms are Nearest Neighbor (NN), Greedy Heuristic (GH), Greedy Randomized Heuristic (GRH), and Greedy Randomized Adaptive Search Procedure (GRASP) [FR95].

Iterated Local Search – In the following, the ILS algorithm is explained in more detail, since it will be used throughout this thesis. ILS is a well-known MH for its effectiveness in both exploration and exploitation, and its simplicity in practice [LMS03]. The general Pseudo code of ILS is given in Algorithm 1 representing that ILS involves iteratively three main operations: *Perturbation*, *Local search*, and *Acceptance*.

Algorithm 1. Pseudo code of the ILS

```
1 get an initial solution s_0
2 s* := LocalSearch(s_0)
3 s_best := s*
4 while termination criterion not reached do
5 |   s' := Perturbation(s*)
6 |   s*' := LocalSearch(s')
7 |   s* := Acceptance(s*, s*', s_best)
8 end
9 return the best found solution s_best
```

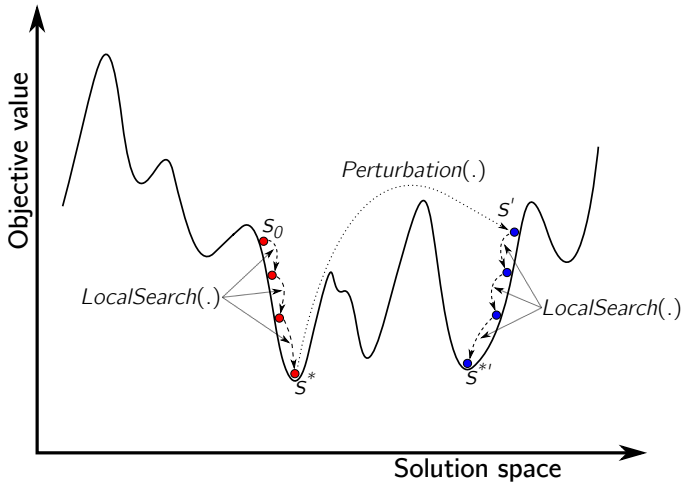

Figure 2 – Iterated local search algorithm

Starting from an initial solution s_0, as in Figure 2, a `LocalSearch(.)` function (i.e., local search operator) is first performed on the initial solution s_0 to defines the starting point s^* for the main loop of the ILS. Subsequently, s^* is archived as the current best solution s_{best}. Each iteration of the main loop involves three consecutive perturbation, local search, and acceptance operations as follows:

- **Perturbation**: ILS performs a `Perturbation(.)` function (i.e., perturbation operator) over the current local optimum solution s^* to help the search process to escape from the local optimum; whereby an intermediate solution s' is generated. The perturbation should not be too severe to act as a random restart but enough powerful to kicks out the solution from the local optimum.
- **Local search**: After applying the `Perturbation(.)` function, the `LocalSearch(.)` function (i.e., local search operator) is performed on the intermediate solution s' to obtain a new local optimal solution $s^{*'}$.
- **Acceptance**: Finally, the `Acceptance`$(s^*, s^{*'}, s_{best})$ function is employed to check whether the new local optimal solution $s^{*'}$ is accepted. The `Acceptance`(.) function can only accept better solution (i.e., *Only Improvement* strategy) or it can even accept worse solution with a small gap (i.e., *Metropolis acceptance* strategy [Met+53]).

These steps are repeated until the termination criterion (e.g., maximum number of iterations, total execution time, number of iterations without improvement, etc.) is satisfied.

2.3 Machine learning

ML is a subfield of artificial intelligence that uses algorithmic and statistical approaches to give computers the ability to "learn" from data, i.e., to improve their performance in solving tasks without being explicitly programmed for each one [Sam88]. The term ”Machine Learning” was coined in 1952 by Arthur Samuel who was a computer scientist at IBM and a pioneer in AI and computer gaming, when he designed a computer program for playing checkers. In fact, the more the program played the game, the more it learned from its experience, thanks to a minimax algorithm for studying moves to find winning strategies. ML is therefore used to teach machines how to handle the data more efficiently or to let algorithms discover patterns in the data. The data can be numbers, words, images, statistics, etc. In fact, anything that can be stored digitally can serve as data for ML. The algorithms improve their learning over time autonomously, using discovered patterns and extracted knowledge from data and information in the form of observations and real-world interactions. The acquired knowledge allows these systems to correctly generalize to new settings.

Accordingly, ML techniques learn from data and build a mathematical model over input data. The input data used to learn ML techniques are usually divided in three data sets: training, validation and test sets [Bis06]. The training dataset is the sample of data used to initially fit the model. In fact, the algorithm sees and learns from this data. The validation dataset is actually used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. Indeed, the algorithm occasionally sees these data, but never learns from them. They are used to update the level of hyperparameters during learning. The test dataset is finally used to provide an unbiased evaluation of a final model fit on the training dataset. Accordingly, these data are only used once the ML technique is completely trained and a final model is fitted (using the train and validation sets).

According to [Bis06], ML techniques can be generally classified into supervised, unsupervised, semi-supervised, and reinforcement learning algorithms. In the following, these classes are explained briefly.

2.3.1 Supervised learning algorithms

In supervised learning, the machine is taught by an example. The operator provides the ML technique with a known dataset, wherein the values of input variables and the corresponding values of the output variables (labels) are known a priori. The algorithm must then automatically figure out the relationship between the input variables and the output labels, and use it to predict the output for new input variables [WBK20]. While the operator knows the correct relation between the inputs and the outputs, the algorithm observes and identifies patterns in data, learns from observations, and makes predictions. While learning, the algorithm makes predictions and is corrected by the operator. This learning process continues until the algorithm achieves a high level of accuracy/performance. Depending on the aim of learning, supervised learning algorithms can be classified into classification, regression, and forecasting algorithms. In classification tasks, the goal is to draw a conclusion from observed values and determine to what class new observations belong. For example, when filtering emails as ”spam” or ”not spam”, the ML technique must observe the available data and filter the emails accordingly. In regression tasks, the ML technique must first understand, then estimate the relationships among a set of independent variables and one dependent variable. In forecasting tasks, the ML technique makes predictions about the future based on the past and present data.

Classical supervised learning algorithms include Linear Regression (LR) to model the relationship between one dependent variable (response) and one or more independent (explanatory) variables, Logistic Regression (LogR) to model the probability of a binary response (e.g., true/false, yes/no) given a set of input variables, Linear Discriminant Analysis (LDA) to find a linear combination of variables (features) that characterizes or separates two or more classes of objects, Support Vector Machine (SVM) to find a hyperplane in an N-dimensional space (N – the number of variables) that distinctly classifies the input data points, Naive Bayes (NB) to classify every variable as independent of any others using probability, Gradient Boosting (GB) to predict a model in the form of an ensemble of weak prediction models, Decision Tree (DT) to create a model that predicts the value of a target variable by learning simple decision rules inferred from the input data, Random Forest (RF) that is similar to DT but operates by constructing a multitude DTs at training time, k-Nearest Neighbor (kNN) to estimate how likely a data point is to be a member of one class or another, and finally, Artificial Neural Network (ANN) that comprises "units" arranged in a series of mutually-connected layers to model non-linear relationships in high-dimensional data, etc [Wit+05].

In a more advanced scheme, Few-Shot Learning (FSL), also known as Low-Shot Learning (LSL) in few sources, is a type of supervised ML technique where the training dataset contains limited information [Wan+20a]. FSL aims to build accurate models with low amount of training data.

2.3.2 Unsupervised learning algorithms

The unsupervised ML techniques are used when there is no label for the training data and no answer key or human operator is available to provide instruction. Therefore, the values of input variables are known while there are no associated value for the output variables. In an unsupervised learning process, the ML technique is then left to figure out and describe the patterns hidden in the input data [WBK20]. Due to more available unlabeled data, an unsupervised ML technique assesses more data and gradually improves its ability to make decisions on that data and becomes more refined.

Clustering, Association Rules (ARs) and Dimension Reduction (DR) are three main tasks of unsupervised learning [WBK20]. In clustering, the main goal is grouping sets of similar data (based on defined criteria) into distinct clusters. Clustering task is useful for segmenting data into several clusters and performing analysis on the data of each cluster to find patterns. In association rules learning, the main aim is to discover interesting relations between variables in large datasets to identify strong rules discovered in the data using some measures of interestingness [KK06]. Dimension reduction is finally to reduces the number of variables being considered to find the exact information required [Cun08].

Classical unsupervised learning algorithms include k-means clustering to categorize unlabeled data into k number of clusters (value of k is given a priori) based on the features provided, Shared Nearest Neighbor Clustering (SNNC) to categorize high-dimensional data into cluster of varying density that share many of their nearest neighbors, Self-Organizing Map (SOM) to produce a low-dimensional (typically two) representation of a higher dimensional dataset while preserving the topological structure of the input data, Principal Component Analysis (PCA) to reduce the dimension of the input data by computing the principal components and using them to perform a change of basis on the data (i.e., keeping the first few principal components and ignoring the rest), Multiple Correspondence Analysis (MCA) to detect and represent underlying structures in nominal categorical data, and finally, Apriori algorithms for ARs to recognize properties

that frequently occur in a dataset and to deduce a categorization [AS+94; WBK20].

2.3.3 Semi-supervised learning

In the two previous learning algorithms, either labels are present for all the observations in the dataset (supervised learning) or there are no labels (unsupervised learning). Semi-supervised learning falls in between the two. In general, semi-supervised learning is similar to supervised learning, but uses both labelled and unlabeled data. The idea behind semi-supervised learning is that even though the group memberships of the unlabeled data are unknown, this data carries important information about the group parameters. By using this combination, ML techniques can then learn to label the unlabeled data. Therefore, the goal of semi-supervised learning is to understand how combining labeled and unlabeled data may change the learning behavior, and design algorithms that take advantage of such combination [ZG09].

2.3.4 Reinforcement learning algorithms

Reinforcement Learning (RL) is a type of ML technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences" [KLM96; SB18]. Figure 3 illustrate a generic scheme of RL algorithm. In RL, the environment is the agent's world in which it lives and interacts. The agent's action is the way by which it interacts with the environment, however; it cannot influence the rules or dynamics of the environment by its actions.

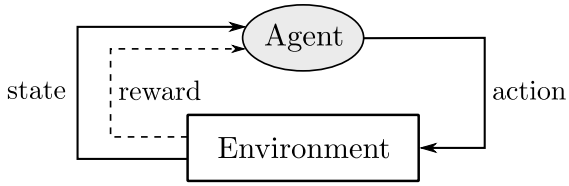

Figure 3 – A generic scheme of RL algorithm

With every interaction of the agent with the environment, the environment returns a new state of the environment, making the agent move to a new state. The environment also sends a positive value (i.e., the so-called *reward*) to the desired actions, and sends a negative value (i.e., the so-called *punishment*) to punish undesired behaviors. These values are scalar values that act as feedback for the agent whether its action was good or bad. This procedure allows the agent to seek long-term and maximum overall reward to achieve an optimal solution. This optimal solution is also called the optimal *policy*. The policy is essentially a probability that tells the agent the odds of certain actions resulting in rewards, or beneficial states. Accordingly, the RL algorithm attempts to iteratively maximize the cumulative reward collected by an agent through trial-and-error interactions with its environment. RL differs from supervised learning in a way that in supervised learning the training data has the labels by which the model is trained whereas in RL, there is no label, and the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, the agent actually learns from its experience.

RL algorithms can be mainly divided into two *model-based* and *model-free* categories [SB18]. In this context, the model helps the agent to infer its environment. For example, using the model, the agent might predict the next state and next reward, given its current state and action. In model-based algorithms, the RL environment can be described with

a Markov decision process (MDP), which consists of the set of states and actions, as well as a reward function and the Transition Probability Distribution (TPD) associated with the MDP. The TPD and the reward function are often collectively called the "model" of the environment. The goal of the agent is then to find out the optimal policy (i.e., the best sequence of actions) using the model of the environment to maximize its cumulative reward. However, for most problems, including COPs, such a model is not available [Wau+13]. In such cases, model-free algorithms can be used that does not use the transition probability distribution.

Among model-free RL algorithms, including Learning Automata (LA), Opposition-based Reinforcement Learning (OPRL), Monte Carlo RL, State-action-reward-state-action (SARSA), Q-learning (QL), SARSA [RN94] and Q-learning [Wat89] are two common algorithms that have been widely used. The SARSA starts by giving the agent what is known as a policy. Accordingly, SARSA is also called an *on-policy* algorithm that learns based on its current action derived from its current policy. The Q-learning, as a Monte Carlo and Temporal Differences algorithm [SB18], is an *off-policy* method in which the agent learns based on an action derived from another policy. Indeed, the agent receives no policy, meaning its exploration of its environment is more self-directed [Wat89]. These techniques actually differ in the way of exploring their environments. Despite their simplicity and freeness of the environment's model, these two algorithms lack generality, as they do not have the ability to estimates unseen states. In the following, the Q-learning algorithm is explained in more detail, since it will be used in our proposed framework in Chapter 4.

Q-learning algorithm – In Q-learning, each state-action pair $(s, a) \in \mathbb{S} \times \mathbb{A}$ is associated with a score $Q(s, a) \in \mathbb{R}$, representing the expected gain associated with the choice of a possible action a at state s:

$$Q(s,a) := Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \,, \tag{2.12}$$

where s is the current state, a is the action taken in state s, s' is the next state after performing action a, and a' is a possible action in state s'. Furthermore, α $(0 < \alpha \leq 1)$ controls the learning rate that determines the ratio of accepting newly learned information, r is the reward received after performing action a, and γ $(0 < \gamma \leq 1)$ is the discount factor that determines the influence of the future reward $\max_{a'} Q(s', a')$. Note that $\mathbb{S}$ is a finite set, and possible actions $\mathbb{A}$ may generally depend on the state.

An important point in the Q-learning algorithm is making a balance between exploration and exploitation when selecting the actions. One strategy could be to always select the action with the maximum Q-value which would augment the exploitation property of the Q-learning algorithm, as the other state-action pairs remain unexplored. On the other hand, Q-learning is proven to converge to the optimal $Q(s, a)$ if each state-action pair is visited numerous times [Wat89]. Therefore, besides the state-action pair with maximum Q-value, other pairs should also be given a chance to be executed/explored. The ϵ-greedy strategy [SB18] is a good strategy that makes a balance between exploration and exploitation by attributing an ϵ selection probability to other actions. It is expressed as follows:

$$a = \begin{cases} \arg\max_{a \in A} Q(s,a) & \text{with probability } 1-\epsilon \\ \text{any action selected uniformly and randomly in } \mathbb{A} & \text{with probability } \epsilon \end{cases} \tag{2.13}$$

It is common practice to reduce the value of ϵ throughout the search process using a parameter β called ϵ-decay. This degradation aims at moving from exploring new actions toward exploiting the best actions throughout the search process.

The Q-learning algorithms becomes less effective if the number of actions and states increase significantly. In such cases, updating the table of Q-value become quickly out of control, and the amount of memory required to save and update the table as well as the amount of time required to explore each state increase unrealistically. In this situation, the Q-values are approximated using other ML techniques like ANNs. These algorithms are called Deep Reinforcement Learning (DRL), or particularly Deep Q-learning (DQL).

2.4 Adaptive Operator Selection & Learning

Through designing MHs with multiple search operators, we can take advantage of several operators with different performances by switching between them during the search process. In addition, an appropriate implementation of different operators significantly affects the exploration and exploitation abilities of a MH and provides an Exploration-Exploitation balance during the search process. In designing such algorithms, one important question to answer is *which operator (among multiple operators) to select and apply at each stage of the search process?* This question is referred to as the *operator selection* problem.

There are two operator selection mechanisms: *offline* operator selection or *online* operator selection. In offline operator selection, the order of the search operators is determined a priori and is kept fixed throughout the search process. On the contrary, in online operator selection, the operators are dynamically selected and employed during the search process. AOS is an online operator selection mechanism that select the operators based on an online feedback on their performance. In this way, AOS gives the chance to adapt MHs behavior to the characteristics of the search space by selecting their operators during the search process based on their historical performance. AOS consists of five main steps: *performance criteria identification*, *reward computation*, *credit assignment*, *selection*, and *move acceptance* which are explained in detail in Section 3.5.1 of Chapter 3.

In AOS literature, most studies have used simple methods for credit assignment. The simple methods assign an initial score (set to a same value, typically zero) to each operator at the beginning of the search and update this score at each iteration by accumulating the scores obtained from the beginning of the search or during limited iterations. The scores could increase by one (additive) if the operator has been successful, and decrease by one (subtractive) if the operator has not been successful [Fia10; Bur+13]. In 2021, a meta-analysis has been done by [TSH21] to investigate the added-value of using simple credit assignment methods, with focus on the adaptive large neighborhood search MH. According to their analysis, AOS with simple credit assignment methods do not bring significant added-value to the overall performance of the MH.

On the other hand, in recent years, the use of ML techniques for AOS has attracted the researchers [MGS20; Ahm+18; GLL18; Zha+21]. ML techniques can help the AOS

to use feedback information on the performance of the operators. To be more specific, ML techniques can be integrated into the *credit assignment* step to gather feedback information from the search, determine how to assign a credit to each operator based on the gathered feedback, and the rules to update this credit throughout the search process. In this regard, reinforcement learning techniques such as Q-learning [MGS20] and learning automata [GLL18] have been used for AOS.

The overall scheme of the AOS framework using ML techniques is shown in Figure 4. This framework consists of two main components, a ML technique integrated into a MH to help select the operators in order to solve a given COP. In this thesis, a general framework is developed based on the idea of Figure 4, wherein we used the Iterated Local Search (ILS) as a MH and the Q-learning algorithm as the ML algorithm.

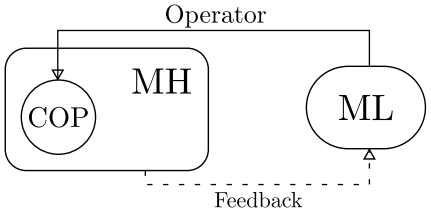

Figure 4 – AOS with an online learning

Chapter 3

Integration of Machine Learning into Meta-heuristics: A Taxonomy

Contents

3.1 Taxonomy and review methodology 46
3.1.1 Contributions 46
3.1.2 Taxonomy 47
3.1.3 Search methodology 48
3.2 Algorithm selection 49
3.2.1 Literature classification & analysis 52
3.2.2 Discussion & future research directions 54
3.3 Fitness evaluation 56
3.3.1 Literature classification & analysis 57
3.3.2 Discussion & future research directions 58
3.4 Initialization 59
3.4.1 Literature classification & analysis 60
3.4.2 Discussion & future research directions 61
3.5 Evolution 62
3.5.1 Operator selection 62
3.5.2 Learnable evolution model 71
3.5.3 Neighbor generation 74
3.6 Parameter setting 76
3.6.1 Literature classification & analysis 77
3.6.2 Discussion & future research directions 78
3.7 Cooperation 79
3.7.1 Literature classification & analysis 80
3.7.2 Discussion & future research directions 82
3.8 Conclusion 82

This chapter presents our first contribution in this thesis, a comprehensive review on the ML-into-MH integration, which has been published in the *European Journal of Operational Research* [Kar]. In this review, first, we propose a unified taxonomy and classification on the use of ML techniques in the design of different elements of MHs for different purposes. According to our classification, ML techniques can be integrated into MHs for different purposes including *Algorithm Selection*, *Fitness Evaluation*, *Initialization*, *Evolution*, *Parameter Setting*, and finally, *Cooperation*. A whole section has been dedicated to each class of the proposed taxonomy. In each section, first, we define the main concepts and classify and review the articles under the corresponding category with focusing on a set of major features (such as the online/offline learning, the used ML techniques, etc.). Then, we analyze the articles in a technical manner, provide technical discussion on the guidelines, requirements, and challenges of this integration, and identify the existing research gaps. Finally, based on the identified gaps, we propose numerous insights for future research.

One of the main contributions of this review that distinguishes it from other review studies on ML-into-MH integration is that this review not only classifies different existing studies in the literature but also follows a technical viewpoint to address a set of the technical challenges that a user may face when integrating ML techniques into MHs. Considering the identified research gaps and the challenges, this review provides a set of research insights and future directions that is worth exploring more.

We take the first step by ourselves to address one of the research gaps identified in this chapter by proposing a general framework to integrate RL into MHs for the purpose of operator selection in Chapter 4.

3.1 Taxonomy and review methodology

This section aims first at providing the main contributions of our review study comparing to the studies existing in the literature. Then it presents a taxonomy to provide a common terminology and classification on the integration of machine learning into meta-heuristics. Finally, the search methodology describing the procedure of searching and obtaining the relevant papers is presented.

3.1.1 Contributions

In this section, we elaborate the main features of our review study that distinguish it from the existing review studies in the literature. There are several review studies in the literature that study the integration of ML techniques and MH, either for a specific purpose or for different purposes. [JDT06] provided a short survey on how ML techniques can help MHs with no detailed discussion on how such integration occurs. Another survey has been done by [Zha+11] on how ML techniques can improve the performance of evolutionary computation algorithms. [CDJ12] investigated the synergy between operations research and data mining, with a focus on multi-objective approaches. With the rapid advances in the use of new ML techniques in MHs for even new purposes, as well as the increasing trend in the number of annually published papers in the area, there is a need to update the outdated review papers [JDT06; Zha+11; CDJ12]. In this regard, [Tal16] studied different ways of hybridization between different MHs as well as hybridizing MHs with mathematical programming, constraint programming, and ML. Although the author provides a good overview on how to hybridize MHs, it is less focused on the integration of ML into MHs. [Cal+17] reviewed the integration of ML and MH for solving optimization problems with dynamic inputs. The authors enumerate different

ways of integrating ML into MH and vice versa; however, their work lacks a technical discussion on the requirements, challenges, and future works of each way of integration.

More recently, more general and comprehensive studies have been done by [STÃ19] and [Tal20] on integrating ML and MH. [STÃ19] studied the integration of ML and optimization in general and not particularly MHs. The authors review all four optimization-in-ML, ML-in-optimization, ML-in-ML, and Opt-in-Opt ways of integration. However, [STÃ19] provided fewer details on the integration of ML in MHs compared to [Tal20]. Merely providing general research directions, it lacks a comprehensive discussion on the research gaps and future research directions for the ML-in-optimization way of integration. [Tal20] provided a more complete and unified taxonomy on the integration of ML into MHs. The author identifies the integration of ML in MHs in three levels: 1) problem level integration, where ML is used, for example, to decompose the solution space or to reformulate the objectives and constraints of an optimization problem, 2) high-level integration between MHs, where ML techniques are used to make a link between different MHs, and 3) low-level integration in a MH, where ML techniques are used in the components of MHs (e.g., initialization, operator selection, population management, etc.). Although the work by [Tal20] is a good comprehensive and pedagogical review paper explaining different general ways (i.e., levels) that ML techniques can be integrated into MHs, it does not go into the details on the requirements, challenges, and possible future research directions on the use of ML techniques in each level of integration.

Accordingly, the main difference of our review comparing to the existing review studies in the literature is that in addition to a classification over all different purposes that ML can be integrated into MHs, this review provides a technical study that identifies and clarifies the technical challenges, requirements, and limitations of this integration. Finally, by providing a broader view over this integration, it proposes new ideas and research directions for future research to fill the identified gaps and deal with the identified challenges.

3.1.2 Taxonomy

Although MHs and ML techniques have been initially developed for different purposes, they may perform common tasks such as feature selection or solving optimization problems. MHs and ML techniques frequently interact to improve their search and/or learning abilities. MHs have been widely employed in ML tasks (MH-in-ML) for decades [WA05; Xue+15]. For Instance, MHs can be used for feature selection, parameter setting of ML techniques [Oli+10], or pattern recognition [KIG14]. ML techniques are being extensively integrated into MHs (ML-in-MH) to make the search process intelligent and more autonomous. For instance, RL can help to select the most efficient operators of MHs during the search process [San+14].

The purpose of this chapter is to review the studies wherein ML techniques have been integrated into MHs for solving COPs. We propose a taxonomy on integrating ML techniques into MHs, ML-in-MH branch of Figure 5. Discovering the MH-in-ML branch of Figure 5 is out of the scope of this thesis, and we refer interested readers to the latest literature reviews on the use of MHs in ML tasks and the references cited therein [Cal+17; STÃ19; GGNS20].

We propose to classify different types of integration according to the taxonomy presented on Figure 5. According to this classification, ML techniques can be integrated into MHs for the following purposes:

- *Algorithm selection* – When solving an optimization problem with MHs, the first

decision is to select one or a set of MHs for solving the problem. ML techniques can predict the performance of MHs in solving optimization problems.

- *Fitness evaluation* – The success of any MH in achieving a specific goal (objective) is evaluated by fitness evaluation of the solutions during the search process. ML techniques can speed up the search process by approximating computationally expensive fitness functions.
- *Initialization* – Any MH starts its search process from an initial solution or a population of solutions. ML techniques can help to generate good initial solutions by using the knowledge of good solutions on similar instances, or speeding up the initialization by decomposing the input data space into smaller sub-spaces.
- *Evolution* – It represents the entire search process, starting from the initial solution (population) toward the final solution (population). ML techniques can intelligently select the search operators (i.e., *Operator selection*), evolve a population of solutions using the knowledge of good and bad solutions during the search (i.e., *Learnable evolution model*), and to guide the neighbor generation process using the knowledge obtained during the search process (i.e., *Neighbor generation*).
- *Parameter setting* – Any MH, depending on its nature, has a set of parameters which need to be set before the search process starts. ML techniques can help to set or control the values of the parameters before or during the search process.
- *Cooperation* – Several MHs can cooperate with each other to solve optimization problems in parallel or sequentially. ML techniques can improve the performance of cooperative MHs by adjusting their behavior during the search process.

Each type of integration in Figure 5 can also be classified from another viewpoint into: *problem-level*, *high-level*, and *low-level* integration [Tal20]. *Algorithm selection* and *fitness evaluation* represent a high-level and problem-level integration of ML into MHs. Depending on the strategy to generate initial solutions, *initialization* belongs to either problem-level or low-level integration. *Evolution* and *parameter setting* fall in the category of low-level integration. Finally, *cooperation* may belong to either high-level or low-level integration, depending on the level of cooperation.

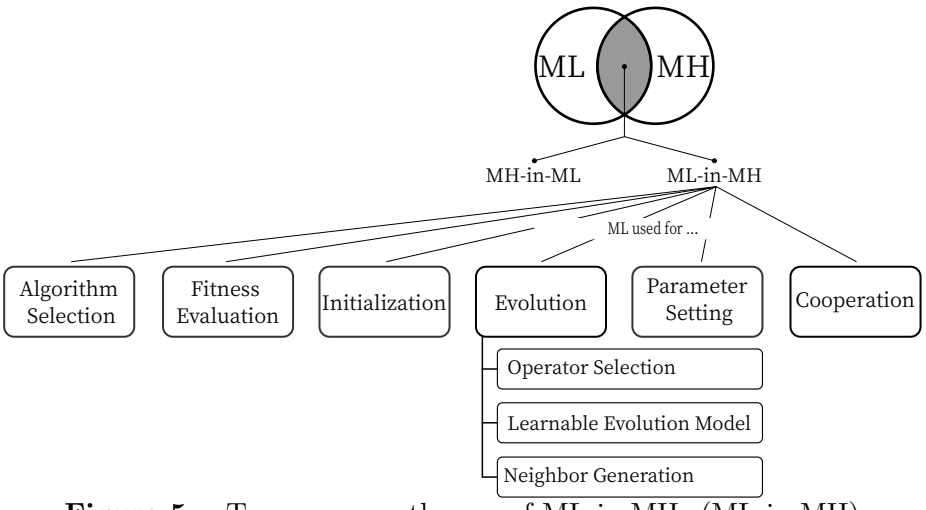

Figure 5 – Taxonomy on the use of ML in MHs (ML-in-MH)

3.1.3 Search methodology

To conduct the literature review, first, well-known scientific databases including Scopus, Google Scholar, IEEE Explore, Science Direct, Springer, ACM Digital Library, and

Emerald have been carefully searched to find the relevant papers in both scientific journals and international conferences. To do that, we have identified a set of particular keywords for each type of integration. Then, the search process is conducted using the following search rule:

{`keyword1` **AND** `keyword2` **AND** `keyword3` **AND** `keyword4`}

`keyword1` is an element of a set of keywords related to the integration types of ML techniques into MHs, as explained in Section 3.1.2 and Figure 5. More precisely, `keyword1` belongs to the union of the sets {algorithm selection, algorithm recommendation, autonomous algorithm selection, performance prediction, meta-learning, meta-feature} (for the algorithm selection), {fitness approximation, surrogate model, metamodel, fitness reduction} (for the fitness evaluation), {initialization, initial solution generation} (for the initialization), {adaptive operator selection, autonomous operator selection, learnable evolution model, non-Darwinian evolution, pattern extraction, rule extraction, rule injection} (for the evolution), {parameter setting, parameter tuning, parameter control} (for the parameter setting), and, {cooperation, cooperative MHs, parallel MHs, Hybrid MHs, sequential MHs} (for the cooperation of algorithms). `keyword2` stands for the ML techniques extracted from Section 2.3 in Chapter 2. `keyword3` accounts for different MHs, ranging from single-solution to population-based MHs, extracted from Section 2.2 in Chapter 2. Finally, `keyword4` is dedicated to the COP under study. We provide a complete list of COPs in Table A.1 in Appendix A.2.

After filtering the obtained papers, a total number of 136 papers are kept, which are relevant to the scope of this review. All these papers are reviewed and classified in details. Figure 6 shows the number of reviewed papers per year (from 2000 to early 2021) and for each type of integration illustrated in Figure 5. Looking at the whole number of papers regardless of which type of integration they belong to, Figure 6 shows a significant increase in the number of papers integrating ML techniques into MHs for different purposes throughout the last two decades, which illustrates a meaningful growth in the knowledge and popularity of the topic. Among all types of integration, studies on *evolution* contribute the most to the total number of papers over time, and an increasing trend can be seen for the last decade. *Algorithm selection* and *initialization* have been at the second place of attention, and they have gained significant attention throughout the last two decades. *Cooperation*, *parameter setting*, and *fitness evaluation* are also the types of integration with semi-constant trend of attention. In summary, Figure 6 illustrates that *algorithm selection*, *evolution*, and *initialization* are being studied attentively, and *fitness evaluation*, *parameter setting*, and *cooperation* are less-studied directions and they are worthy to be explored more in the future.

The rest of this chapter is structured as follows. Each section explains in details each way of integrating ML techniques into MHs and starts with an introduction to the corresponding type of integration. Then, relevant papers are reviewed, classified, and analyzed. Finally, the chapter ends with a comprehensive discussion on the corresponding guidelines, requirements, challenges, and future research directions. Finally, conclusions and perspectives are given in Section 3.8.

3.2 Algorithm selection

There are many studies in the literature developing high-performance MHs for well-known COPs. However, there is no single MH that dominates all other MHs in solving all problem instances. Instead, different MHs perform well on different problem instances (i.e., performance complementarity phenomena) [Ker+19]. Therefore, there is

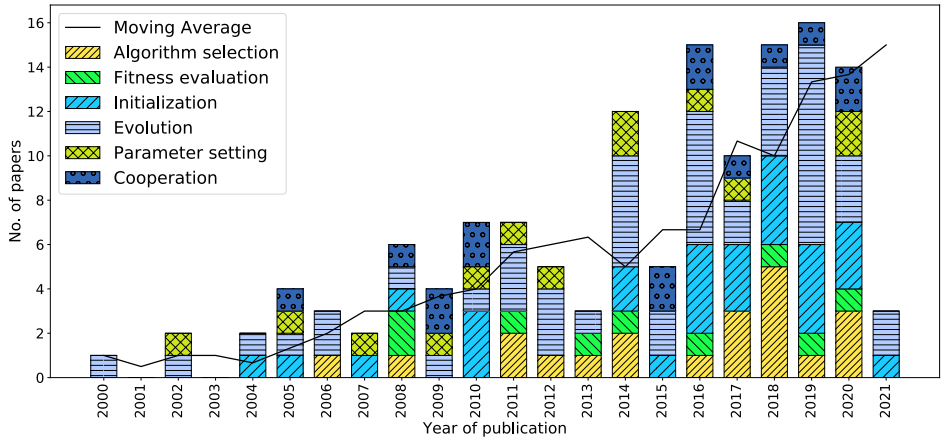

Figure 6 – Number of papers per year and per each type of integration of ML techniques in MHs for solving COPs

always an unsolved question as "Which algorithm is likely to perform best for a given COP" [Ric+76]? The ideal way to find the best algorithm to solve a COP, when the computational resources are unlimited, is to exhaustively run all available algorithms and choose the best solution, no matter by which algorithm it has been obtained. However, because of the limited computational resources, it is practically impossible to test all available algorithms on a particular problem instance. In this situation, a major question arises as "Among the existing algorithms, how to select the most appropriate one for solving a particular problem instance?". ML techniques help to answer this question by selecting the most appropriate algorithm(s). This is where the Algorithm Selection Problem (ASP) steps in.

ASP aims at automatically selecting the most appropriate algorithm(s) for solving a problem instance using ML techniques [Ker+19; Kot14]. The original framework of ASP was developed by [Ric+76] based on four principal components: 1) the problem space, including a set of problem instances, 2) the feature space, including a set of quantitative characteristics of the problem instances, 3) the algorithm space, including a set of all available algorithms for solving the problem instances, and 4) the performance space that maps each algorithm from the algorithm space to a set of performance metrics such as the Objective Function Value (OFV), CPU Time (CT), etc. The final goal is to find the problem-algorithm mapping with the highest performance.

To find the best problem-algorithm mapping, ASP employs Meta-learning, a subfield of ML, that learns the problem-algorithm mapping on a set of training instances and creates a metamodel. The metamodel is then used to predict the appropriate mapping for new problem instances [Kot14]. In solving COPs, studying ASP has enabled researchers to take advantage of various MHs by systematically selecting the most appropriate algorithm among the existing ones, and has resulted in significant performance improvements [Kot16]. In the literature, ASP has been referred to as *algorithm selection* [Kan+16], *per-instance algorithm selection* [Ker+19], *algorithm recommendation model* [Chu+19], and *automated algorithm selection* [DP18]. However, they all share the same goal of automatically selecting the most appropriate algorithm(s) for a particular problem instance.

We illustrate the procedure of ASP in Figure 7. As it can be seen in Figure 7, ASP involves two main steps: 1) meta-data extraction, and 2) meta-learning and metamodel creation.

- **Meta-data extraction** – Given the problem and algorithm spaces, the goal is

to determine the feature and performance spaces called meta-data. Meta-data is classified into two categories: meta-features and meta-target features [Kan+16]. Meta-features are a set of quantitative features that represent the properties of a problem instance, while meta-target features are a set of performance data that describe the performance of each algorithm on a particular problem instance. Considering the importance of defining appropriate meta-features, there are several works in the literature which aim to identify good meta-features for different COPs, including SAT [Ker+19], TSP [SML12; Mer+13; Ker+19], AP [AZ02; SML12], OP [Bos+18], KP [SML12], BPP [SML12], and GCP [SML12].

- **Meta-learning and metamodel creation** – Using the meta-data, a metamodel is created which can predict the performance of each algorithm for each problem instance and determine the problem-algorithm mapping. Depending on the type of prediction expected from the metamodel, different ML techniques can be used for meta-learning and creating the metamodel. Different types of prediction include selecting the best algorithm [SM08], selecting a set of most appropriate algorithms [Kan+11a], and ranking a set of appropriate algorithms [Kan+16] for solving a problem instance. Depending on the type of prediction, the task of meta-learning could be *Single-label Classification* (SLC), *Multi-label Classification* (MLC), and *Label-ranking Classification* (LRC), respectively. Besides classification techniques, regression techniques such as LR can also be used to predict the performance of each MH. Using regression techniques, the meta-learning problem is a multiple regression problem wherein one target variable is considered for each MH.

Considering the way to create the metamodel, ASP can be either *online* or *offline*. In offline ASP, the metamodel is constructed using a particular set of training instances with the aim to predict the problem-algorithm mapping for new problem instances [SM08; Kan+16; Mir+18]. However, in online ASP, the metamodel is constructed and employed dynamically while solving a set of problem instances [Arm+06; GS10; Deg+16; Ker+19]. Furthermore, the algorithm space, commonly known as algorithm portfolio, is classified as *static* or *dynamic*. Static portfolios contain a set of fixed algorithms which are included into the portfolio before solving a problem instance and the composition of the portfolio along with the algorithms within the portfolio do not change during solving an instance, while dynamic portfolios contain a set of algorithms whose composition and configuration may change while solving a problem instance [Kot14].

In the rest of this section, the research papers studying ASP for COPs are reviewed and classified, followed by a detailed discussion on the corresponding guidelines, requirements, challenges, and future research directions.

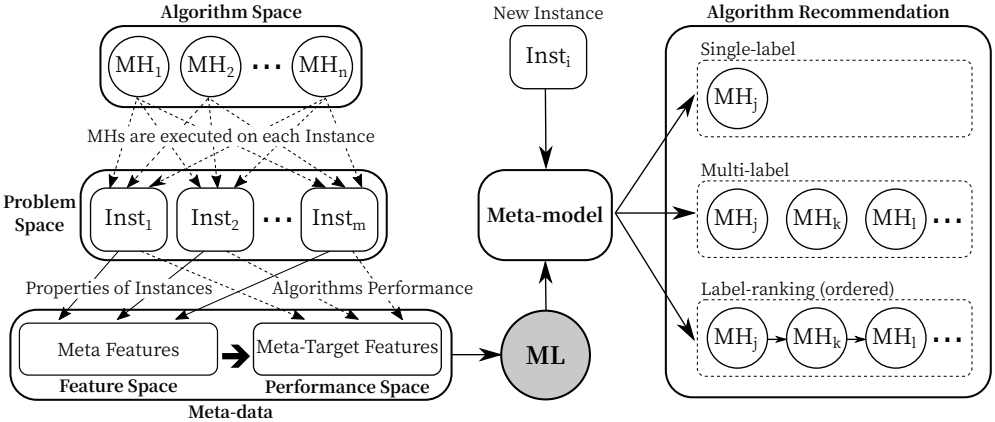

Figure 7 – Procedure of ASP

3.2.1 Literature classification & analysis

Inspired from Figure 7, Table 2 classifies the papers studying ASP for COPs based on different characteristics such as *problem space*, *algorithm space*, algorithm *portfolio* type, *performance space*, *learning* mechanism, meta-learning *task*, the employed *ML technique*, and the *size* of the training set. To the best of our knowledge, Table 2 lists all relevant papers, including the most recent papers in the literature that study ASP for selecting MHs to solve COPs using ML techniques. Other papers which miss at least one of these three main components (i.e., MHs, ML techniques, or COPs) are out of the scope of this thesis and are not reviewed in this thesis.

Regarding the problem space, TSP and QAP are the most studied problems compared to the other COPs. Also, Table 2 shows that a majority of the studied COPs have a common characteristic, they either have permutation-based representation (e.g., TSP, VRP, FSP) or discrete value based representation (e.g., AP, QAP). One of the major reasons for such an observation is the availability of various powerful MHs for these representations, as well as the simplicity of manipulating these types of representations [Koç+16; AA16; AB+18].

Considering the algorithm space, Table 2 reveals that the algorithm portfolio in most of the studies is composed of MHs with different mechanisms, varying from single-solution to population-based, from memory-less MHs (e.g., ILS) to MHs with memory (e.g., TS), from MHs with accepting only better solutions (e.g., ILS) to MHs with accepting worse solutions (e.g., SA), and from MHs with fixed neighborhood size (e.g., TS and SA) to MHs with variable neighborhood size (e.g., VNS). The utilization of such different algorithms in a portfolio highlights the fact that different MHs with different search mechanisms perform differently for different instances of COPs [San+14].

Table 2 also shows that all reviewed papers use a static portfolio. Since the algorithms and their configurations do not change in static portfolios, their selection becomes more crucial for the overall success of the resolution process. An efficient way to construct the portfolio is to involve algorithms that complement each other, such that good performance can be achieved on a wide range of different problem instances. There has been a debate on the composition and characteristics of the algorithms within the portfolio among the reviewed papers. The first and the most straightforward manner to construct the portfolio is to randomly select algorithms from a large pool of diverse MHs. The second manner is to incorporate MHs with the best overall performance in the portfolio. However, the third and the most promising manner is to construct a portfolio with algorithms of complementary strengths. An ASP with a portfolio composed of MHs with complementary strengths logically seems to be more efficient, comparing to an ASP with a portfolio composed of MHs with the best overall performance. However, most research papers construct the portfolio less explicitly using the MHs that have performed well in the literature when solving particular instances of the COP at hand, regardless of their strengths and weaknesses when facing new problem instances.

Table 2 – Classification of papers studying ASP

Ref.	Prob. space	Alg. space	Portfolio	Perf. space	Learning	Task	ML tech.	Size
[Hut+06]	SAT	ILS	Static	CT	Offline	Reg.	LR	30000
[SM08]	QAP	ILS, TS, ACO	Static	OFV	Offline	SLC	ANN	644
[Kan+11a]	TSP	TS, GRASP, SA, GA	Static	OFV	Offline	MLC	k-NN, ANN, NB	2500
[Kan+11b]	TSP	TS, GRASP, SA, GA	Static	OFV	Offline	LRC	ANN	2000
[Kan+12]	TSP	TS, SA, GA, ACO	Static	OFV	Offline	LRC	ANN	300
[PBA13]	QAP	TS, VNS	Static	OFV	Offline	SLC	LR, SVM, AR	137
[MDC14]	PSP	TS, GA	Static	OFV	Offline	SLC	DT	3140
[SM+14]	GCP	HC, TS, ACO	Static	OFV	Offline	MLC	SVM, NB	675

To be continued ...

Table 2 (continued)

Ref.	Prob. space	Alg. space	Portfolio	Perf. space	Learning	Task	ML tech.	Size
[Kan+16]	TSP	TS, SA, GA, ACO	Static	OFV	Offline	LRC	ANN, k-NN, DT	600
[BAW17]	QAP	TS, VNS, GA, MA	Static	OFV, CT	Offline	SLC	k-NN	94
[Leó+17a]	BAP	LNS	Static	OFV	Offline	LRC	k-NN	720
[Leó+17b]	TSP, VRP	GRASP, SA, LNS	Static	OFV	Offline	LRC	k-NN	130
[Mir+18]	MAX-SAT	GA, PSO	Static	OFV	Offline	SLC	ANN, SVM, DT	555
[PKD18]	FSP	HC, SA, TS, ILS	Static	OFV	Offline	MLC	GB	27000
[PDK18]	FSP	HC, SA, TS, ILS	Static	OFV	Offline	SLC	DT	12000
[DP18]	QAP	BLS, ACO, TS	Static	OFV	Offline	SLC	RF	135
[DGVDC18]	AP	TS	Static	OFV	Offline	SLC	RF	286
[GR+19]	VRP	EA, GA, PSO	Static	OFV	Offline	SLC	ANN	56
[DP20]	QAP	BLS, ACO, MA	Static	OFV	Offline	MLC	RF	5000
[WDS20]	BAP	HC, GRASP, ILS	Static	OFV, CT	Offline	SLC	k-NN	2100
[Sad+20]	MAX-SAT	GA, GRASP	Static	OFV, CT	Offline	SLC	k-NN, RF, ANN	1534
[RR+21]	TTP	ILS, SA, VNS	Static	OFV	Offline	Reg.	LR	6000

Considering the *performance space*, we can see that most of the papers evaluate the performance of an algorithm based on the OFV of the obtained solutions. Although considering the quality of solutions in terms of their OFV is the most common criterion to compare algorithms, there are other criteria that play an important role when selecting an algorithm, among which the CT and robustness have a high importance, especially for solving COPs [San+14; CWL19; MGS20]. Therefore, OFV, CT, and robustness are the three most important criteria by which the algorithms could be compared. Although there is a trade-off between these measures, and usually no algorithm performs best in all criteria, taking into account these criteria provides more efficient algorithm selection when solving COPs [BAW17; WDS20]. The multiple criteria ASP can be modeled through a multi-objective perspective [Ker+19].

Table 2 shows that all reviewed papers have created the metamodel in an offline manner, and none of them studies the online ASP for solving COPs. A big disadvantage of ASP in an offline manner is that in this way, the performance of the selected algorithms is not monitored to confirm whether they satisfy the expectations that led them being selected or not. Accordingly, offline ASP is inherently vulnerable to bad choices of MHs; however, the advantage of an offline ASP is its lower computational effort since the metamodel is created once based on a set of training instances. On the contrary, the major advantage of an online ASP is the more justified decisions that can be made during the algorithm selection process, which also reduces the negative impact of a bad choice. However, adding such flexibility imposes an extra effort, as the metamodel is created and employed dynamically while solving a set of problem instances and thus decisions on algorithm selection need to be made more frequently throughout the resolution of the new problem instances. Broadly speaking, there is no evidence to show the superiority of one method over the other, and both methods have led to performance improvements [Kan+16; WDS20]. Hence, the choice of whether to create the metamodel in an offline or an online manner depends highly on the specific application.

Another studied characteristic in Table 2 is the meta-learning task. The most common output of ASP is a single best algorithm from the portfolio and using it to solve the problem instance (i.e., the result of SLC task). A disadvantage of selecting a single best algorithm is having no way of compensating a wrong selection. Indeed, if a single algorithm is selected and shows unsatisfying performance on a new problem instance, there are no other recommended algorithms to replace such an inefficient algorithm. An alternative approach is selecting multiple algorithms (i.e., the result of MLC and RLC tasks). However, there is no report to show that one of these approaches is superior to another.

3.2.2 Discussion & future research directions

In this section, first, a guideline is provided for researchers on when to study ASP and which requirements to meet to study ASP. Second, a set of technical challenges of studying ASP are discussed. Finally, several future research directions are provided based on the research gaps extracted from Table 2.

3.2.2.1 Guideline & requirements

The aim of providing a guideline for studying ASP is to help researchers to understand that although studying ASP may provide the most appropriate MH(s) to solve COPs, it is not always the best choice. In the following, we first describe the situations where ASP is useful; then, the requirements of using ASP are elaborated.

Studying ASP is useful when the computational resources (i.e., available time and the number of available cores) for solving a problem instance are limited. This is the case for optimization problems at an operational level where limited time is available, and the problems should be solved more frequently. On the other hand, for the optimization problems at the strategic level (e.g., FLP), where there is enough time, the best choice is to execute all algorithms and select the most appropriate one, since in the strategic level, finding better solutions outweighs the computational cost of executing all algorithms. Furthermore, another moment when studying ASP becomes indispensable is when there are several efficient competitive algorithms for the problem at hand and none of them could be definitely selected for solving the problem instance. In addition, ASP can help non-experts to select appropriate algorithm(s) for solving optimization problems. In other words, ASP can be replaced by the traditional trial-and-error optimization tasks, especially when the number of candidate algorithms is large and little prior knowledge of the problem is available.

Once the use of ASP is justified, a set of requirements should be fulfilled before applying ASP. The first requirement for using ASP is *affordability* of the selection procedure in terms of computational resources. In fact, if studying ASP for a problem instance is more expensive than solving the problem instance with all algorithms and selecting the best one, there is no need at all to study ASP. The next important requirement that could be also a challenge for ASP is *data availability*. When creating the metamodel, it is necessary to provide a pool of sufficient training instances that well represent new instances. It should be noted that having a pool of sufficient instances does not guarantee the efficiency of ASP, and *instance dissimilarity* and *algorithmic discrimination* are two other requirements that need to be satisfied [SM+14]. The former denotes the necessity of providing instances which are as diverse as possible and spread out over different regions. The latter denotes the necessity to provide instances that show different behaviors while being solved by different algorithms in the portfolio. Indeed, some instances should be easy for some algorithms and hard for others. *Algorithmic discrimination* requirement helps to learn the strengths and weaknesses of different algorithms when solving different instances with different characteristics.

3.2.2.2 Challenges & future research directions

Despite the effectiveness of ASP in solving COPs, the implementation of an algorithm selection procedure is not always straightforward, and researchers may face several challenges throughout the ASP procedure, from the design to its implementation.

The first challenge to deal with is called *data generation* challenge. As mentioned earlier, two important requirements of ASP are *instance dissimilarity* and *algorithmic discrim-*

ination that allow generating a rich data set of instances with different characteristics that leads to a more efficient metamodel creation. To generate such a rich data set, the algorithms need to be provided with a wide range of instances. This challenge is twofold: 1) Finding or generating a set of sufficient instances that ensure the *data availability* requirements, particularly *instance dissimilarity* and *algorithmic discrimination*, is a complicated task and 2) Executing all candidate algorithms on the generated instances might be very time-consuming if the number of instances is large. This makes the metamodel creation computationally expensive. This first challenge becomes more and more complicated if little knowledge is available for the COPs at hand. On the other hand, it would be less challenging to generate a set of sufficient instances that fulfils the mentioned requirements for classical COPs for which there exists several instance libraries (e.g., TSPLIB for TSP [Rei91] and Taillard for FSP [Tai93]).

Apart from the *data generation* challenge, the second challenge is *instance characterization*. A major issue in creating a metamodel is the characterization of the problem instances through a set of appropriate measures, called meta-features [SML12]. Meta-features must reveal instance properties that affect the performance of the algorithm. More informative and appropriate features lead to a better mapping between the meta-features and algorithm performance, and consequently a high-quality metamodel. The *instance characterization* challenge has two aspects; first, the type of meta-features to extract and, second, the computational time associated with the meta-feature extraction. The type of meta-features varies from the most basic ones such as descriptive statistics (e.g., minimum, maximum, mean, and median of input parameters) to more complex ones such as landscape features of COPs. Taking TSP as an example, the basic meta-features include "Edge and vertex measures" such as *number of vertices*, *the lowest/highest vertex cost*, and *the lowest/highest edge cost*, and the more complex meta-features could be "complex network measures" such as *average geodesic distance*, *network vulnerability*, and *target entropy* [Kan+16]. Depending on the type of meta-features, feature extraction can be a computationally cheap task for basic features and expensive for more complex ones. Therefore, in selecting the meta-features, one should consider both the level of information they provide and their corresponding computational time. The optimal way is to select a set of features that are as informative as possible while computationally affordable. To put the issue into perspective, creating a high-quality metamodel is a complicated interplay between using a set of diverse training instances with different behavior over different algorithms and using a subset of informative meta-features whose extraction is computationally cheap.

There is always an unanswered question on the trade-off between the performance of the algorithm selection and its complexity, particularly on the extraction of meta-features. An important direction for future research is moving from problem-specific features toward more general and simple features, which are computationally cheaper to be extracted when studying ASP. There is evidence that shows for particular optimization problems, a small number of simple meta-features suffice for achieving excellent performance of ASP [Hoo+18].

In the reviewed studies, the focus has been on heterogeneous portfolios composed of different MHs with different characteristics; while different configurations of a single MH in a homogeneous portfolio also show different behavior in solving a COP. Another intriguing future research direction could be studying ASP for COPs with a homogeneous portfolio, to select a particular configuration of a single MH among a set of particular configurations, which is even less challenging compared to dealing with a portfolio of different algorithms.

As explained in Section 3.2.2.1, *instance dissimilarity* and *algorithmic discrimination* are two requirements and also two challenges of ASP. An interesting research direction could be the idea of evolving instances using EAs [Hem06; SMHL10; Mer+13; BT16]. Indeed, the idea is to use EAs to evolve instances of COPs as distinct as possible to ensure instance dissimilarity. In this way, a set of diverse instances are obtained, improving the performance of the metamodel.

Developing an online ASP is another promising future research direction. As shown in section 3.2.1, all papers have studied offline ASP, where the metamodel is created using a set of training instances. However, it might be possible to get even better results using online ASP, which adapts the algorithm selection mechanism while solving a set of problem instances. Although the online ASP imposes an extra computational overhead, it increases the robustness, as adjustments (if required) can be applied to the algorithm selection mechanism during the resolution of COP instances. It is worth mentioning that the extra overhead can be alleviated by using computationally cheap meta-features. Another way to cope with the extra overhead is using incremental or online active learning techniques, where an already trained model is used and improved incrementally during the search process [Lug17].

When the output of ASP is multiple selected algorithms, one can study how multiple selected algorithms are *scheduled* to solve a problem instance. The key idea of scheduling is to execute a sequence of algorithms from a given set, one after another, each for a given (maximum) time [Kot14]. Most of the research papers in Table 2 proposing multiple algorithms have not studied the algorithm schedule. However, more flexibility is obtained throughout the search process when an algorithm with particular strength (i.e., exploration and exploitation) is employed whenever needed. Accordingly, one future research direction could be scheduling multiple recommended algorithms to solve a problem instance. The schedule of algorithms can be either *static* or *dynamic* [Kad+11]. In a *static* algorithm schedule, algorithms are executed based on a given order. In a *dynamic* schedule, the sequence of algorithms may change based on their historical performance, and certain algorithms may not be used at all.

Another interesting future research direction could be taking into account multiple performance criteria by which the algorithms are evaluated when studying ASP. The multiple criteria ASP can be modeled through a multi-objective perspective [Ker+19]. Last but not least, Table 2 reveals that the focus has been mostly on COPs with permutation-based and discrete value based representations. As a future research direction, ASP can be extended to other COPs such as different types of scheduling problems for which a lot of efficient MHs have been developed in recent years [All15].

3.3 Fitness evaluation

Fitness evaluation is one of the key components of MHs to guide the search process towards the promising regions of the search space. For some optimization problems, there is no analytical fitness function by which the solutions are evaluated, or even if it exists, it is computationally expensive to evaluate. ML techniques can be integrated into MHs to reduce the computational effort for solving such optimization problems either through *fitness approximation* [Jin05; Jin11; DMTC17] or *fitness reduction* [Sax+12]. Fitness approximation is categorized into *functional approximation* and *evolutionary approximation* [Jin05]:

- **Functional approximation** – It is used when evaluating the solutions using the original fitness function is computationally expensive. In this condition, the com-

putationally expensive fitness function is replaced with an approximate model that imitates the behavior of the original fitness function as closely as possible, while being computationally cheaper to evaluate. These approximate models are built using ML techniques such as polynomial regression [SRS10], RF [Zho+05], ANN [JS04; PK17], SVM [LSS10; GJAP19], Radial Basis Functions (RBFs) [Qas+13], and Gaussian process models also referred to as Kriging [Kno06]. These ML techniques are trained using a set of training data, wherein the input variable is a set of features extracted from the solution instances and the output variable is the original fitness value of each solution instance. The aim is then to approximate the fitness value of the new generated solutions. The approximate model can be created either *offline* or *online*. A particular use of functional approximation in MHs is known as surrogate-assisted MHs [Jin11], wherein the approximate (surrogate) model is iteratively refined (i.e., online refining) during the search process. The first surrogate-assisted MHs were developed for continuous optimization problems, and there are numerous efficient surrogate modeling techniques for continuous functions [Pel+20]. In recent years, they have also gained attraction for discrete optimization problems [BBZ17]. Surrogate models are categorized into single, multi-fidelity, and ensemble surrogate models [BBZ17].

- **Evolutionary approximation** – It is specifically developed to deal with EAs. Instead of approximating the fitness function, the evolutionary approximation aims at reducing the computational effort by approximating the elements of the EAs. There are two main sub-categories of evolutionary approximation:
  - *Fitness inheritance* in which the fitness value of an individual is calculated based on the fitness values of its parents. For instance, the fitness value of an offspring can be the (weighted) average of the fitness values of its parents.
  - *Fitness imitation* in which the fitness value of an individual is calculated using the fitness value of its siblings. Using ML techniques such as clustering techniques, the population is divided into several clusters, and only the representative individuals of each cluster are evaluated. Afterward, the fitness values of other individuals are calculated based on their corresponding representatives in the clusters. Clustering techniques have been widely used for fitness imitation in the literature [Yu+17; Xia+20].

Apart from fitness approximation, *fitness reduction* is another approach for dealing with computationally expensive fitness functions in multi-objective optimization problems [Sax+12]. Instead of approximating the fitness function, fitness reduction aims at reducing the number of fitness functions using ML techniques such as PCA [Sax+12] and Feature Selection (FS) techniques [LJCCC08] as well as reducing the number of fitness function evaluations by using clustering techniques [Zha+16b; Sun+19].

In the following, first, we review, classify, and analyze the relevant papers, and then the corresponding challenges and future research directions are provided.

3.3.1 Literature classification & analysis

Table 3 classifies the papers using ML techniques for fitness evaluation of COPs based on different characteristics such as the *fitness evaluation* approach, the type of the problem (*single/ multi objective*), the employed *ML technique*, the *MH* algorithm, and the *COP* under study.

Table 3 – Classification of papers studying fitness evaluation

Ref.	Fitness evaluation	Single/ Multi obj.	ML tech.	MH	COP
[PSS08]	Functional approximation	Multi	ANN	GA	PSP
[LJCCC08]	Fitness reduction	Multi	FS	GA	KP
[MKY11]	Functional approximation	Single	RBF	GA	QAP
[Hor+13]	Functional approximation	Single	ANN	MA	ATOP
[Ngu+14]	Evolutionary approximation	Single	k-NN	GA	JSP
[Hao+16]	Functional approximation	Single	ANN	DE	SMSP
[WJ18; ZFX19]	Functional approximation	Multi	RF	GA	KP
[Luc+20]	Functional approximation	Single	DT	VNS	VRP

Considering Table 3, it can be seen that there are few studies applying fitness approximation/ reduction to COPs. The reason is twofold; first, for most COPs, there exists an analytical fitness function whose evaluation is not computationally expensive [HL14], and second, constructing surrogate models for COPs is a complicated task and several additional issues should be overcome to create an accurate and reliable surrogate model for COPs [Pel+20].

It can be seen from Table 3 that fitness approximation is mostly used for single-objective COPs whose original fitness function is calculated by a time-consuming simulation [Hor+13; Hao+16], or an approximate fitness function is used to help the search to escape from the local optima [Luc+20]. There are also studies that apply fitness approximation to multi-objective COPs. Indeed, it is more computationally expensive for a MH to evaluate solutions using multiple fitness functions compared to a single fitness function, especially when there are too many objective functions.

Table 3 shows that fitness approximation/ reduction is mostly used for EAs (e.g., GA and MA). The main reason is that EAs generate and evolve a population of solutions at each iteration, and it might be therefore very time-consuming to evaluate every new solution at each iteration of the algorithm. Using fitness approximation especially becomes crucial when the evaluation of each new solution is computationally expensive (e.g., using time-consuming simulation to evaluate a solution [Hor+13]). This necessity has led to the development of new EAs called surrogate-assisted EAs.

3.3.2 Discussion & future research directions

As all MHs do an iterative process to reach the (near-) optimal solution, many fitness evaluations are needed to find an acceptable solution. Fitness approximation may help MHs to significantly reduce their computational effort for computing the fitness value [Jin05]. However, using fitness approximation in MHs is not as straightforward as one may expect, and it has its own challenges.

One of the major challenges is the accuracy of the approximate function and its functionality over the global search space [Jin05]. To replace the original fitness function of a MH with an approximate function, it has to be ensured that the MH with the approximate function converges to the (near-) optimal solution of the original function. However, due to some issues such as few training data and high dimensionality of the search space, it is difficult to construct such an approximate function. Therefore, to overcome this issue, one way is to use both the original and the approximate fitness function during the resolution of the problem. This is addressed as model management or evolution control in the literature [Jin05].

An open question in fitness approximation is choosing the best suited technique for fitness approximation in COPs. The answer mainly depends on the COP under study and the user's preferences; however, due to numerous approximation techniques, selecting the best suited technique a priori is often impossible. In this regard, the first try

could be using the simplest technique. If the performance of the approximate function obtained by the simplest technique is unsatisfactory or degrades over time, more sophisticated techniques can be used. A future research direction could provide a comparative study on the performance of different techniques for approximating the fitness function of COPs. These techniques may differ from simple techniques such as fitness inheritance or k-NN to more sophisticated ones such as polynomial regression, Kriging, RBF, and clustering/classification techniques [SR10]. Usually more complex methods provide better fitting accuracy but need more construction time. Another way to answer this open question is using *ensemble surrogate modeling* that aggregates several surrogate models.

Apart from fitness approximation, another aspect that deserves to be explored more in the future is online fitness generation, wherein new objectives are targeted depending on the status of the search process. The new fitness function is generated based on some knowledge of the optimization problem at hand, as well as the features extracted from the visited regions during the search process. For instance, a set of representative features of the good solutions for the COP at hand can be extracted during the search process to form a new objective, and once the MH gets trapped in a local optimum, the original fitness function is replaced by the new one. During the search process, the original and the new fitness function can interchange to guide the MH toward promising solutions [Luc+20]. Another example of online fitness generation in MHs can be found in GLS that modifies the original fitness function when trapped in a local optimum [VT03]. Such a modification is done by adding a set of penalty terms to the original fitness function. Whenever the GLS gets stuck in a local optimum, the penalties are modified and the search process continues to optimize the transformed fitness function.

Real-time COPs are those COPs that need to be solved regularly (e.g., every hour or every day) under a time limitation. For these problems, the computational time spent even in one iteration of MHs, especially population-based MHs, may be too long for real-time applications. Therefore, to cope with real-time COPs, especially large-scale COPs, one future research direction is to employ fitness approximation to lower the computational effort of fitness evaluation.

3.4 Initialization

There are three main strategies for generating initial solutions for MHs: *random*, *greedy*, and *hybrid* strategies [Tal09]. In the random strategy, an initial solution is generated randomly, regardless of the quality of the solution. In the greedy strategy, a solution is initialized with a good-enough quality. Finally, the hybrid strategy combines two random and greedy strategies. There is always a trade-off between the use of these strategies in terms of exploration and exploitation. Indeed, the way of initializing a MH has a profound impact on its exploration and exploitation abilities. If the initial solutions are not well diversified, a premature convergence may occur, and the MH gets stuck in local optima. On the other hand, starting from low quality solutions may take a larger number of iterations to converge. In this regard, ML techniques can be used not only to maintain the diversity of solutions but also to produce initial solutions with good quality. ML techniques can contribute to the initialization through three major strategies:

- **Complete generation** – As a low-level integration, ML techniques can replace the solution generation strategies to construct the initial solution on their own. Indeed, ML techniques are used to construct a complete solution from an empty solution. The main ML techniques used in this category are RL-based techniques

such as QL [San+14; Kha+17] , ANN [Ben+20], and Opposition-based Learning [RTS08].

- **Partial generation** – As a low-level integration, ML techniques are used to generate partial initial solutions using the apriori knowledge of good solutions. Then, the remaining part of the solution can be generated using any of the initialization strategies. ML techniques extract knowledge from previous good solutions and inject it into the new initial solutions [LO05; Nas+19]. This knowledge is mostly in the form of ARs, which characterize the properties of good solutions. Apriori algorithms are widely used to extract the rules in ARs [Li+16]. Another example is case-based initialization strategy [LM04] derived from the idea of Case-Based Reasoning (CBR), in which the initial solutions are generated based on the solutions of already solved similar instances.
- **Decomposition** – As a problem-level integration, the decomposition is done either in the data space or in the search space. In data space decomposition, ML techniques are used to decompose the data space into several sub-spaces and consequently facilitate generating initial solutions by reducing the required computational effort. In this regard, an initial solution is generated for each subspace using any of the initialization strategies, and finally a complete solution is constructed from the partial solutions of the sub-spaces [Cha17; MJL19; AEK20]. In search space decomposition, ML techniques are used to diversify the initial solutions over the search space, where different solutions represent different regions of the search space. For example, the problem of selecting the subregions of the search space to explore can be formulated using the Multi-armed Bandit (MAB) technique, wherein each arm represents a region of the search space, and the technique learns which regions worth exploring further and which are not [CDM14].

Depending on how ML techniques are employed in generating the initial solutions, learning can occur either *offline* or *online*. In offline learning, knowledge is gathered from the initial solutions generated for a set of training instances with the aim to generate initial solution(s) for a new problem instance. The properties of those good initial solutions that led to a better performance are extracted and used to generate promising initial solution(s) for a new problem instance. Although an offline learning can provide rich knowledge, it might be very time-consuming, and the extracted knowledge might not be useful enough when applied to a new problem instance with completely different properties compared to the training instances. On the contrary, in online learning, knowledge is extracted and employed dynamically while generating the initial solution(s) for a problem instance. Although the extracted knowledge might not be that rich, it completely suits the instance at hand. In the rest of this section, the relevant papers are reviewed and analyzed and the corresponding challenges along with future research directions are provided.

3.4.1 Literature classification & analysis

Table 4 classifies the papers generating initial solutions for COPs using ML techniques based on different characteristics such as the initialization *strategy*, *learning* mechanism, the used *ML technique*, the *MH* algorithm for which the initialization is performed, the *COP* under study, and the *size* of the training set in case of offline learning.

Considering Table 4, RL, particularly QL is a widely used technique to generate complete initial solutions. QL can be counted as a hybrid initialization strategy that balances exploration and exploitation abilities of a MH through its parameters. QL constructs the solutions successively by exploiting the knowledge of the search space using the

reward matrix. Taking TSP as an example, QL starts construction with a random city and proceeds with the cities which bring the maximum reward, where the reward is representative of the problem's objective function (e.g., the reward is inversely related to the distance from the current selected city to the next potential city). The superiority of QL over other typical initialization strategies in terms of the quality of the solutions and convergence rate of the algorithm has been illustrated for TSP by [San+14].

Table 4 shows that the decomposition strategy is mostly used for routing-based COPs including VRP and TSP. Clustering algorithms such as k-means are the widely used ML techniques in these studies, where the cities are clustered into several groups, and an initial solution is obtained by using a greedy strategy for each group. Finally, a complete path connecting different groups is created.

3.4.2 Discussion & future research directions

In the integration of ML techniques into MHs for generating initial solutions, the simplicity of the classical random and greedy strategies is sacrificed to gain better performance in terms of the trade-off between exploration and exploitation through more advanced initialization strategies (i.e., complete generation, partial generation and decomposition). The integration of ML techniques into initialization of MHs has been reported to lead to an improvement in the convergence rate of MHs when better solutions have been found in less computational efforts compared to classical random and greedy strategies [LM04; San+14; Has+18; AEK20]. These improvements have been more expressive when solving larger instances of COPs as MHs start with already a (set of) good initial solution(s) [San+14; Has+18]. This phenomenon saves the computational effort toward exploring/exploiting more promising regions in the solution space, instead of spending extensive efforts to find primary good (local) solutions during the search process.

Table 4 – Classification of papers studying initialization

Ref.	Strategy	Learning	ML tech.	MH	COP	Size
[LM04]	Partial generation	Offline	Apriori	GA	JSP	50
[LO05]	Partial generation	Offline	DT	MHs	JSP	19900
[LJMN07]; [DLDMN08]; [San+10]	Complete generation	Online	QL	GRASP, GA	TSP	–
[DPRVZD10]; [HZG10]	Decomposition	Online	k-means	GA	VRP	–
[San+14]	Complete generation	Online	QL	VNS	TSP	–
[CDM14]	Decomposition	Online	MAB	HC	QAP	–
[DLZ15]	Decomposition	Online	k-means	GA	TSP	–
[ZHD16]	Complete generation	Online	RL	DLS	GCP	–
[Li+16]	Partial generation	Online	Apriori	GA	TSP	–
[XP16]; [Zha17]	Decomposition	Online	k-means	ACO	VRP	–
[Gao+16]	Decomposition	Online	k-means	ACO	LRP	–
[Kha+17]	Complete generation	Online	QL	MHs	TSP	–
[Cha17]	Decomposition	Online	k-means	ACO	TSP	–
[Has+18]	Decomposition	Online	LR	GA	TSP	–
[LSRVMG18]	Decomposition	Online	k-means	ACO	WSRP	–
[Ali+18]	Complete generation	Online	RL	GA	TSP	–
[MYE18]	Partial generation	Offline	DRL	MHs	TSP	200000
[AEK19]	Decomposition	Online	k-means	GA, DE	TSP	–
[GY19]	Decomposition	Online	k-means	GA	VRP	–
[MJL19]	Decomposition	Online	k-means	TS	VRP	–
[Nas+19]	Partial generation	Offline	Apriori	GA, PSO	JSP	35
[Ben+20]	Complete generation	Offline	LR, ANN	MHs	FLP	45000
[LMR20]	Partial generation	Offline	LogR, ANN, NB	MHs	FLP	7145
[AEK20]	Decomposition	Online	k-means	DE	TSP	–
[Che+21]	Decomposition	Online	k-means	ABC	JSP	–

These advanced initialization strategies bring their own challenges. An important challenge is the complexity of using such advanced techniques with additional parameters that need to be carefully tuned to get the highest performance. A set of other challenges

arises depending on the way solutions are initialized. For instance, a big challenge when using QL is how to define the set of states and actions so that they satisfy the properties of a Markov decision process. One of the main requirements is to define a set of states such that it is sufficient to characterize the system without the need for the history of information achieved so far. Taking TSP as an example, if one considers the state as the currently selected city and the action as the next city to be added to the tour, the state is not sufficient to characterize the system, and an action depends on the information on the history of the states.

As a new concept in ML and inspired from the opposite relationship among entities, Opposition-based Learning can provide efficient strategies to generate the initial population. Using this concept, the initial population is approximated from two opposite sides [RTS08], wherein an initial population is first generated randomly. Next, an opposite population in terms of values of the solution vector is generated to the randomly generated population. The MH then merges the two populations and selects half of the best solutions to form the initial population. The main aim of opposition-based learning is keeping the diversity between the initial solutions to increase the exploration ability of the MHs. In addition to opposition-based learning, an interpolation technique can also be used to generate the initial population. This technique attempts to provide good solutions by interpolating a set of randomly generated solutions. The interpolated solutions are then used to form the initial population. Neither opposition-based learning nor interpolation have been applied to COPs. Therefore, using these techniques and other advanced ML techniques such as ANNs [YA01] to generate the initial solutions of COPs could be a future research direction.

3.5 Evolution

ML techniques can be integrated into the evolution process in three major ways:

- ML techniques help to use feedback information on the performance of the operators during the search process to select the most appropriate operator (see Section 3.5.1).
- ML techniques provide a learning mode to generate new populations in EAs (see Section 3.5.2).
- ML techniques help to extract the properties of good solutions to generate new solutions (see Section 3.5.3).

3.5.1 Operator selection

Operator selection has its roots in *hyper-heuristics*. The term *hyper-heuristic* can be defined as a high-level automated search methodology which explores a search space of low-level heuristics (i.e., neighborhood or search operators) or heuristic components, to solve optimization problems [Bur+13]. Regarding the nature of the heuristic search space, hyper-heuristics are classified into *heuristic selection* and *heuristic generation* methodologies [Dra+19]. The former aims at selecting among a set of heuristics, while the latter aims at generating new heuristics. Operator selection inherently belongs to *heuristic selection* methodologies in hyper-heuristics; however, it has been also used in designing MHs [San+14; MGS20]. Accordingly, it has led to a certain level of confusion in the literature in distinguishing MHs involving operator selection from hyper-heuristics. Despite the differences between the design and performance of hyper-heuristics and MHs, operator selection in both methods targets the same goal as selecting and applying (an) appropriate operator(s) during the search process. In this work, the focus is on MHs

involving operator selection. In this regard, a MH called adaptive large neighborhood search, an extension to the classical large neighborhood search, has been developed with the particular aim of selecting operators during its search process. A meta-analysis on adaptive large neighborhood search MHs has been provided by [TSH20]. Apart from adaptive large neighborhood search, this work focuses on operator selection in all other types of MHs.

Search operators are divided into four categories [Tal09; OB14]:

- **Mutational/ Perturbation Operators (MPOs)** - They are unary operators that perform small changes on a single individual solution by swapping, changing, reversing, inserting, or removing solution components. The aim of the mutation operator is thus to explore the neighborhood of the current solution, or to roam the undiscovered regions of the search space.
- **Ruin-Recreate (destruction-construction) Operators (RROs)** - These operators partially ruin a solution and then rebuild or recreate it. Differing from MPOs, these operators can be considered as large neighborhood structures that can incorporate problem specific construction operators to rebuild the solution.
- **Local Search or Hill-Climbing Operators (LSOs)** - These operators iteratively make small changes to a solution, by only accepting non-deteriorating changes, until a local optimum is found, or a stopping condition is met. Differing from MPOs, these operators perform an iterative improvement process and guarantee that a non-deteriorating solution will be produced. However, this condition can be relaxed if deteriorating solutions are also accepted in order to escape the local optima (e.g., SA).
- **Crossover or Recombination Operators (XROs)** - Unlike unary operators, the crossover operators are binary and sometimes n-ary that take two solutions (called "parents" in EAs), combine them and return either a single or multiple new solutions (called "offspring"). The role of the crossover operators is thus to inherit the characteristics of two solutions to generate new solutions.

AOS, as an online operator selection mechanism, consists of five main steps:

- **Performance criteria identification** – Whenever an operator is selected and applied to a problem instance, a set of feedback information can be collected that represents the performance of the algorithm. This feedback can be different performance criteria such as OFV, Diversity of Solutions (DOS), CT, and Depth of the Local Optima (DLP). The credit of an operator highly depends on how the performance criteria are identified and assessed. This makes performance criteria identification an important step in AOS. Therefore, in solving COPs, the performance criteria should be efficiently identified and integrated (in case of multiple criteria) to lead the MH toward the optimal solution.
- **Reward computation** – Once the performance criteria are identified, this step computes how much the application of each operator improves/deteriorates the performance criteria.
- **Credit assignment (CA)** – In this step, a credit is assigned to an operator based on the rewards calculated in the reward computation step. There are different credit assignment methods including Score-based CA (SCA) [Pen+19], Q-Learning based CA (QLCA) [Wau+13], Compass CA (CCA) [Mat+09], Learning Automata based CA (LACA) [GLL18], Average CA (ACA) [Fia10], and Extreme Value-based CA (EVCA) [Fia10] presented in Table 5.

Table 5 – Credit assignment methods

Method	Description
SCA	As a simple version of RL, it assigns an initial score (credit) to each operator and updates their credits based on their performance at each step of the search process. Generally, initial credits are set to a same value, typically zero. $$C_{i,t+1} = C_{i,t} + r_{i,t}$$ where $C_{i,t}$ and $r_{i,t}$ are the credit and the reward of operator i at time (step) t.
QLCA	It assigns a Q-value (credit) to an operator (action) at each state of the search process based on its previous performance. $$Q(s_t, a) = Q(s_t, a) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a)]$$ where s_t is the state at time t, s_{t+1} is the new state at time $t+1$, a is the operator selected in state s, a' is a possible operator in state s_{t+1}, r_t is the reward (punishment) received after selecting operator a at time t, α is the learning rate, and γ is the discount factor which indicates the importance of future rewards.
CCA	It integrates three measures at every application of an operator: population diversity variation, mean fitness variation, and execution time. A sliding window stores the last W changes in terms of *diversity* and *fitness*. A compromised value is then calculated between the diversity and fitness measures. Finally, the compromised value is divided by the operatorâĂŹs execution time to obtain the final credit of each operator.
LACA	It assigns a probability value to each operator based on its previous performance. The following formulations show the selection probabilities (credits) of successful and unsuccessful operators, respectively. $$p_{i,t+1} = p_{i,t} + \lambda_1 r_{i,t}(1 - p_{i,t}) - \lambda_2(1 - r_{i,t})p_{i,t}$$ $$p_{i,t+1} = p_{i,t} - \lambda_1 r_{i,t} p_{i,t} + \lambda_2(1 - r_{i,t})[(K-1)^{-1} - p_{i,t}]$$ where $p_{i,t}$ is the selection probability of operator i at time t, λ_1 and λ_2 refer to the learning rates used to update the selection probabilities, and K is the number of operators.
ACA	It assigns credit to each operator according to its performance achieved by its last W applications (Instantaneous credit assignment if W=1). Using W as the size of the sliding window for each operator, the performance of an operator is aggregated over a given time period.
EVCA	It follows the principle that infrequent, yet large improvements in the performance criteria are likely to be more effective than frequent but moderate improvements. At each application of an operator, the changes in the performance criteria are added to a sliding window of size W following a FIFO rule and the maximum value within the window is assigned as a credit to that operator. Despite the ACA, this method emphasizes on rewards to the operators with recent large improvements even once throughout their last W applications. $$C_{i,t} = \max_{t'=t-W,\ldots,t} \{r_{i,t'}\}$$

- **Selection** – Once a credit has been assigned to each operator, AOS selects the operator to apply in the next iteration. Different selection methods including Random selection (RS), Max-Credit Selection (MCS), Roulette-wheel Selection (RWS) [GLL18], Probability Matching Selection (PMS) [Fia+08], Adaptive Pursuit Selection (APS) [Fia+08], Soft-Max Selection (SMS) [GB17], Upper Confidence Bound Multi-Armed Bandit Selection (UCB-MABS) [Fia+08], Dynamic Multi-Armed Bandit Selection (D-MABS) [Mat+09], Epsilon Greedy Selection (EGS) [San+14], and Heuristic-based Selection (HS) [CWL18] are presented in Table 6.
- **Move acceptance** – After the application of an operator, AOS decides whether to accept the move provided by the operator or not. Different move acceptance methods including All Moves Acceptance (AMA), Only Improvement Acceptance (OIA), Naive Acceptance (NA), Threshold Acceptance (TA), Metropolis Acceptance (MA) [Met+53], Probabilistic Worse Acceptance (PWA), Simulated Annealing Acceptance (SAA) [MGS20], and Late Acceptance (LTA) are listed in Table 7.

Table 6 – Selection methods

Method	Description
RS	It uniformly selects an operator at random, ignoring the credit values. $$p_{i,t} = \frac{1}{K}$$ where K is the number of operators.
MCS	It selects an operator with the maximum credit.
RWS	It assigns a selection probability $p_{i,t}$ to operator i at time t based on its proportional credit, and selects an operator randomly based on these probabilities. The more the credit of an operator, the more the chance to be selected. $$p_{i,t+1} = \frac{C_{i,t}}{\sum_{j=1}^{K} C_{j,t}}$$ where $C_{i,t}$ is the assigned credit of operator i at time t, and K is the number of operators.
PMS	It assigns a selection probability to each operator based on its proportional credit, while keeping a minimum selection probability for all operators to give them a chance to be selected regardless of their credit. $$p_{i,t+1} = p_{min} + (1 - K \times p_{min}) \frac{C_{i,t}}{\sum_{j=1}^{K} C_{j,t}}$$ where p_{min} is the minimum selection probability of each operator, K is the number of operators, and $C_{i,t}$ is the assigned credit of operator i at time t.
APS	Instead of proportionally assigning probabilities to all operators, it selects the operator with the maximum credit, increases its probability, and reduces the probabilities of all other operators. Operator i^* is selected as follow: $$\begin{cases} p_{i,t+1} = p_{i,t} + \beta(1 - (K-1)p_{min} - p_{i,t})(\beta > 0) & i^* = arg\ \max\{C_{i,t}\} \\ p_{i,t+1} = p_{i,t} + \beta(p_{min} - p_{i,t}) & \text{otherwise} \end{cases}$$ where the notations are similar to PMS and β is the learning rate.
SMS	It uses a Boltzmann distribution to transform the credit of each operator to a probability, and involves a temperature parameter τ to amplify or condense the differences between the operator probabilities. It uniformly selects an operator based on the probability values. As long as the temperature decreases, this method becomes more greedy towards selecting the best available operator. $$p_{i,t+1} = \frac{e^{(C_{i,t}/\tau)}}{\sum_{j=1}^{K} e^{(C_{j,t}/\tau)}}$$
UCB-MABS	It assigns a cumulative credit to each operator and selects the operator with the maximum value of: $$C_{i,t} + G\sqrt{\frac{\log \sum_{j=1}^{K} n_{j,t}}{n_{i,t}}}$$ where $n_{i,t}$ denotes the number of times the ith arm has been played up to time t and G is the *Scaling* factor used to properly balance rewards and application frequency (Exploration-Exploitation balance) while still maintaining a small selection probability of other operators for exploration purposes.
D-MABS	It adapts the classical multi-armed bandit scenario to a dynamic context where the reward probability of each arm is neither independent nor fixed. To address the dynamic context, the classical UCB [ACBF02] algorithm is combined with a Page Hinkley test [Pag54], to identify the change of reward probabilities.
EGS	It selects the operator with the highest credit with probability of $(1 - \epsilon)$; otherwise, it selects an operator randomly. $$i^* = \begin{cases} arg\ \max_j C_{j,t} & \text{with probability} \quad 1 - \epsilon \\ \text{any other operator} & \text{with probability} \quad \epsilon \end{cases}$$
HS	It uses either heuristic rules or optimization algorithms to select the operators. A heuristic rule can be a tabu list of operators to exclude them from being selected during a certain number of iterations. On the other hand, the sequence of operators can be defined as a decision variable and optimization algorithms (e.g., MHs) are employed to find the optimal sequence.

ML techniques help the operator selection to use feedback information on the performance of the operators. In this situation, operators are selected based on a credit assigned to each operator (i.e., feedback from their historical performance). Considering the nature of the feedback, the learning can be *offline* or *online*. In offline learning, ML techniques such as case-based reasoning [BPQ06] help to gather knowledge in the form of rules from a set of training instances with the aim to select operators in new problem instances. However, in online learning, knowledge is extracted and employed dynamically while solving a problem instance [Tal16; Bur+19].

Table 7 – Move acceptance methods

Method	Description
AMA	It always accepts the applied move regardless of whether the move improves the solution or not.
OIA	It only accepts the moves that improve the solution.
NA	It always accepts the moves that improve the solution and considers an acceptance probability of 0.5 for moves that deteriorate the solution.
TA	It always accepts the moves that improve the solution and accepts the moves that deteriorate the solution less than a prefixed threshold (in terms of the solution quality).
MA	It accepts each move with a probability of $e^{\frac{\Delta f}{T}}$, where Δf is the difference between the fitness values before and after applying that move, and T denotes the temperature. The higher the value of T, the higher the chance to accept worse moves and vice versa.
PWA	As a variant of Metropolis acceptance, it accepts each move with a probability of $e^{\frac{\Delta f}{T*\mu_{impr}}}$, where μ_{impr} is the average of previous improvements on the solution quality.
SAA	As a variant of Metropolis acceptance, it accepts each move with a probability $e^{(\frac{\Delta f}{T*\mu_{impr}} \times \frac{t_{max}}{t_{max}-t_{current}})}$, wherein the temperature T decreases gradually over time. In addition, t_{max} denotes the maximum time allowed to execute the algorithm, and $t_{current}$ denotes the current elapsed time.
LTA	It accepts a move that provides a solution with better or equal quality compared to the obtained solutions in the last n iterations. During the initial n iterations, any move that provides a better solution compared to the initial solution is accepted.

3.5.1.1 Literature classification & analysis

Table 8 classifies the papers studying AOS for COPs based on different characteristics such as *performance criteria*, *credit assignment* method, *selection* method, *move acceptance* method, the *MH* algorithm, type of the *operator*s to be selected, and the *COP* under study.

Considering the performance criteria, Table 8 indicates that all reviewed papers rely on OFV as the criterion used for evaluating the operators. In the meantime, only few papers have incorporated other criteria such as CT and DOS into their evaluation process [MS08; Mat+09; Sak+10; ST12; DT+15]. Although OFV is the most straightforward criterion by which the operators can be evaluated, DOS is also needed to avoid premature convergence [DT+15].

By looking at Table 8, it can be seen that among the credit assignment methods, RL-based methods (e.g., simple SCA and QLCA) have been mostly studied. Among the studied credit assignment methods, the ACA method is biased toward conservative strategies. Indeed, using this method, the operators with frequent small improvements are preferred over operators with rare but high improvements. Despite ACA method, EVCA method assumes that rare but high improvements are even more important than frequent but moderate ones [Fia+08]. Using the EVCA method, the operators are credited based on the maximum improvement during their last W applications. In order to avoid the premature convergence, CCA method has incorporated DOS into its evaluation process. This method takes into consideration both the OFV and DOS, which are related to the exploitation and exploration abilities of MHs, respectively. The ACA, EVCA, and CCA methods assign credit to the operators based on their immediate performance (through the last W applications). This may rise the possibility that the optimization is short-sighted.

On the other hand, RL-based methods including SCA, QLCA, and LACA methods assign credit to the operators based on their performance from their very first application. Indeed, they are able to learn a policy to maximize the rewards in a long-term prospect, which makes it possible to gain optimal operator selection policies. In addition to a long-term prospect, some RL-based methods such as QL are model-free that do not require the complete model of the system including the matrix of transition probabilities and the expected value of the reward for each state-action pair.

Table 8 – Classification of papers studying AOS

Ref.	Perf. criteria	Credit asnt.	Selection	Move acpt.	MH	Operator	COP
[PE02]	OFV	SCA	MS	OIA	GA	MPO, XRO	TSP
[MS08]	OFV, DOS, CT	CCA	PMS	OIA	GA	MPO, XRO	SAT
[Mat+09]	OFV, DOS, CT	CCA, EVCA	D-MABS	OIA	GA	MPO, XRO	SAT
[Sak+10]; [ST12]	OFV, CT	QLCA	SMS	OIA	GA	MPO, XRO	TSP
[Fra+11]	OFV	EVCA	APS, PMS	OIA	MA	XRO	QAP
[Bur+11]	OFV,FT	EVCA	APS, RWS	AMA	ILS	MPO, XRO, RRO, LSO	FSP
[Wal+12]	OFV	EVCA	APS	AMA-OIA	ILS	MPO, XRO, RRO, LSO	VRP
[Han+14]	OFV	QLCA	PMS, APS	OIA	MA	XRO	QAP
[San+14]	OFV	QLCA	EGS	OIA	VNS	LSO	TSP
[CY14]	OFV, DOS	CCA	D-MABS	OIA	MA	XRO	ARP
[BKB14]	OFV	QLCA	EGS, SMS	OIA	EA	MPO, XRO	TSP
[Yua+14]	OFV	SCA	APS, D-MABS	OIA	MA	XRO	QAP
[DT+15]	OFV, DOS, CT	CCA	PMS	OIA	EA	XRO	SAT
[Li+15]	OFV	ACA	RWS	OIA, AMA	ILS	MPO, LSO	VRP
[LT16]	OFV	SCA	RWS	OIA	VNS	LSO	VRP
[SFH16]	OFV	QLCA	APS	OIA	EA	MPO, XRO	TEPP
[Moh+16]	OFV	SCA	RWS	OIA	ICA	XRO	HLP
[Che+16]	OFV	SCA	UCB-MABS	OIA	VNS	LSO	VRP
[Zha+16a]	OFV	SCA	RWS	OIA	GA	MPO, XRO	LRP
[GB17]	OFV	ACA	UCB-MABS	SAA	ILS	MPO	NRP
[GLL18]	OFV	LACA	RS, RWS	OIA, SAA	ILS	LSO	OP
[Ahm+18]	OFV, DOS	QLCA	EEGS	OIA	GA	MPO, XRO	SSP
[MJTM19]	OFV	SCA	RWS	OIA	GA	MPO, XRO	HLP
[Pen+19]	OFV	SCA	RWS	OIA	MA	LSO	VRP
[LZY19]	OFV	SCA	RWS	OIA	ILS	MPO, LSO	VRP
[MGS20]	OFV	QLCA	EGS	SAA	SA	LSO	MASP
[Zha+21]	OFV	QLCA	EGS	OIA	WWO	MPO	FSP
[KM+21]	OFV	QLCA	EGS	OIA	ILS	LSO	TSP

This is especially useful for COPs since generally, a complete model is not available for COPs [Wau+13]. Furthermore, many RL-based methods are able to converge to the optimal state-action pair under several conditions. Among these methods, QL has proven to converge to the optimal state-action pair under three conditions: the system model is a Markov decision process, each state-action pair is visited many times, and the immediate reward given to each action is not unbounded (i.e., limited to some constant) [Wat89]. Among the RL-based methods in Table 8, the techniques based on Temporal Differences such as QL take advantage of the concept of delayed reward. They are based on the assumption that there might be a delay before the effect of an action appears. Accordingly, they consider a delayed reward in addition to an immediate reward. On the other hand, LACA method [GLL18] and SCA method [Che+16] work in a single state environment and merely take into account an immediate reward.

The selection step decides which operator to apply in the next iteration. This step can also be seen as an exploration and exploitation dilemma, where there is a need to be a trade-off between selecting the best operator with the best performance so far (i.e., exploitation), and giving a chance to the other operators which may bring the best performance from then (i.e., exploration). Figure 8 illustrates how different selection methods balance exploration and exploitation abilities of a MH. For each selection method, the responsible parameters and their corresponding effects for making such balance have been also identified. For example, in APS method, increasing the parameters p_{min} and β augments the exploration ability of the method, while decreasing p_{min} and β increases the exploitation ability of the method.

As it can be seen in Figure 8, the MCS method is a purely exploitation-based method where there is no chance for exploring other operators. In other words, the MCS method selects the operator with the maximum credit at each iteration, giving no chance to other lower quality operators to be selected.

The RWS method selects the operators based on their proportional credit, where there is also a chance for all operators to be selected. However, using the RWS method, operators

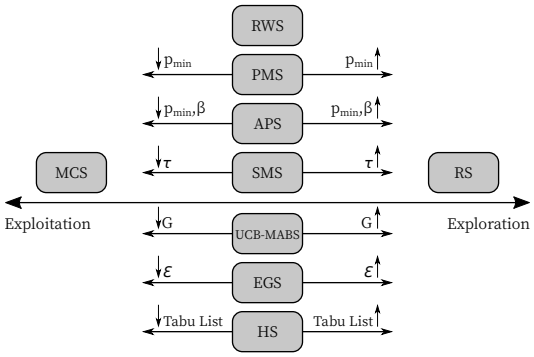

Figure 8 – Exploration vs. Exploitation of selection methods

which do not show good performance during a long time, have very low or even zero chance to be selected, while they might perform well in the latter stages of the search process. To tackle this issue, the PMS method assigns a minimum selection probability to each operator, p_{min}, regardless of its performance. This preserves the balance between exploration and exploitation thorough a minimal level of exploration that is kept fixed during the search process. Indeed, the selection probability of operators with zero credit slowly converges to p_{min}. In this way, the operators with even moderate performance keep being selected, and this degrades the performance of this method [Mat+09]. To address this drawback, the proposed APS method updates the selection probabilities using the winner-takes-all strategy. Indeed, instead of updating the probabilities based on operators' proportional credit, the APS method increases the selection probability of the best operator while decreasing the selection probability of all other operators. This aims at quickly enhancing the application probability of the current best operator [Fia10]. Considering the trade-off between exploration and exploitation, this method keeps a minimal level of exploration through p_{min}, which is fixed throughout the search process.

In the SMS method, the balance between exploration and exploitation is controlled through a temperature parameter, τ. As the temperature increases, the selection probabilities tend to be equal for all operators. While decreasing the temperature leads to a larger difference between selection probabilities. In an extreme viewpoint, when the temperature goes to zero, SMS becomes a purely exploitation-based method where only the best operator is selected. The UCB-MABS method also makes a trade-off between the exploration and exploitation abilities of a MH by keeping a minimum selection probability for each operator to be selected through $G\sqrt{\frac{\log \sum_{j=1}^{K} n_{j,t}}{n_{i,t}}}$, where G is a scaling factor to balance the trade-off. Similarly, the EGS method controls the trade-off between exploration and exploitation using the predefined parameter ϵ. Increasing ϵ increases the exploration ability of the algorithm by giving a chance to the other operators to be selected, while decreasing ϵ favors the selection of the best operator. Accordingly, when $\epsilon = 1$, the EGS method becomes a purely exploration-based method. Finally, the HS method uses a heuristic rule to select operators. For instance, the rule could be a tabu list of operators that excludes successful operators from being selected during a certain number of iterations. The size of the tabu list makes a balance between exploration and exploitation. As the tabu size increases, the successful operators remain longer in the tabu list, giving a chance to other operators to be selected, which leads the method toward exploration.

Table 8 indicates that AOS is mostly applied to EAs (e.g., GA and MA) to select the mutational and crossover operators. The reason may rely on the popularity of EAs for solving COPs and the availability of a variety of problem-specific mutational and

crossover operators, which needs to be carefully selected during the search process since the performance of EAs is highly affected by its operators. In the meanwhile, AOS is also applied to select the local search operators in ILS and VNS.

3.5.1.2 Discussion & future research directions

In this section, first, a guideline is provided for researchers on when to use AOS, and the fundamental requirements of using AOS are identified. Next, challenges and future research directions are discussed.

There is a rise in the number and variety of problem-specific operators (heuristics) for efficiently solving optimization problems. Selecting and applying these operators within a MH requires much expertise in the domain. That is especially the case for COPs with plenty of proposed problem-specific operators, where the classical operators are not as competent as problem-specific ones. Indeed, for COPs with standard representations (e.g., permutation-based representations), users can make the use of classical non-specialized operators, which does not necessarily require much expertise. However, for COPs out of this standard framework, one must have knowledge over the problem-specific operators to efficiently select the operators. This issue highlights the necessity of an automatic approach to select the most appropriate operator(s) based on their performance without having an expertise in the domain. In this way, even inexperienced users are able to select appropriate operators for solving COPs. In addition, AOS is most useful when dealing with several competitive state-of-the-art operators for solving a COP such that none of them can be preferred over the others a priori, and choosing the best operator exhaustively is computationally expensive. In this condition, there is a need for automatic operator selection.

There are a set of specific requirements for the QLCA method as the most common RL-based method used in AOS. Before applying the QLCA method in AOS, one needs to define the set of possible states and actions. In defining the states, the following preconditions should be checked:

- The states should be completely descriptive of the problem status to allow selecting the correct action. There are three ways to define the states. The states could be 1) search-dependent that reflect the properties of the search process such as the number of non-improving iterations, 2) problem-dependent that reflect the properties of the problem through generic features, or 3) instance-dependent that reflect the properties of the problem instance such as the number of bins in a bin packing problem [Wau+13].
- The states should be defined in such a way that do not grow exponentially and allow the algorithm to visit each state-action pair many times. This is one of the main conditions of QLCA method convergence. For instance, if the number of states grows exponentially with the size of the problem, the algorithm might need to be executed more times to satisfy the convergence condition.

The integration of ML techniques into AOS has led to an improvement in terms of both solution quality and even computational time. Using advanced ML techniques in AOS such as QLCA [Ahm+18; MGS20; Zha+21] and LACA [GLL18] has brought significant improvements compared to the non-learning version of MHs. Besides, even new best-known solutions have been obtained for certain COPs [Ahm+18; GLL18]. More interestingly, such integration has been reported to be more efficient as the size of the problem instances increases, and the proposed MHs have shown more stable behavior when solving larger COP instances [San+14; Zha+21].

Apart from the advantages that AOS brings into application, a set of important challenges arises in this regard. The first challenge is related to the computational overhead of learning in the MHs. Although AOS gives the user the flexibility to adapt the MH's behavior to the characteristics of the search space by selecting its operators during the search process, achieving such flexibility adds an extra computational overhead. Keeping track of the performance of the operators during the search process, assigning credits to them, and updating their selection probabilities all impose an extra computational overhead. This overhead can be compensated by an optimal design of the AOS mechanism wherein the MH converges sooner to the (near-) optimal solution, and consequently the saved computational time compensates the extra overhead.

Another challenge is related to the tuning of the parameters of AOS. Most credit assignment and selection methods introduce new parameters that need to be tuned before applying AOS. Tuning the values of these parameters can significantly affect the performance of AOS; thus they need to be carefully tuned. For example, in the QLCA method, there are two new parameters, the learning rate (α) and the discount factor (γ). The former controls the ratio of accepting the newly learned information, while the latter controls the impact of the future reward. Higher levels of α tend to the replacement of the old information by new information. On the other hand, lower values of α emphasize on the existing information. Furthermore, as γ increases, more emphasis is given to future reward compared to the immediate reward.

This challenge becomes more critical when tuning a set of parameters responsible for making a balance between the exploration and exploitation abilities of MHs since they directly control the behavior of MHs and influence the performance of AOS. One way to overcome this challenge is to use the parameter setting methods explained in Section 3.6, wherein the parameters of the MHs are tuned offline or controlled in an online manner. In almost all papers so far, these parameters are considered fixed during the search process, while they can be dynamically adjusted based on the characteristics of the search space. Accordingly, employing an online parameter control method (see Section 3.6) within AOS can be a promising future research direction.

Another challenge in AOS is related to the number of operators involved in AOS. Increasing the number of operators may reduce the performance of AOS. As the number of operators increases, more effort is required to perform AOS, and consequently a higher level of overhead is imposed on AOS. In addition, the performance of AOS may degrade if some operators do not perform well, and they will be selected fewer and fewer in the long term. The presence of such operators increases the computational overhead with no significant gain. One way to overcome this challenge is to use Adaptive Operator Management (AOM) that aims to manage operators during the search process by excluding inefficient operators and including other operators in AOS [Mat+11]. The use of AOM in AOS could be further investigated as a future research direction.

Another promising research direction could be employing AOS in multi-objective COPs where several objectives are evaluated simultaneously. An issue in this regard is how to assign a reward to an operator that improves one objective function but degrades the other objectives. In addition, integrating multiple rewards into a single credit value to be assigned to each operator is another issue to be addressed. One way to cope with these issues could be incorporating the crowding distance as well as the rank of the non-dominated fronts to calculate the reward/credit.

3.5.2 Learnable evolution model

Darwinian-type EAs (e.g., GA) are inspired from the principles of Darwin's theory of evolution. They apply usual genetic operators like mutation, crossover, and selection to generate new populations. These semi-random operators, which govern the evolution process, do not consider the experiences of individual solutions, the experience of an entire population, or a collection of populations. Therefore, new solutions are generated through a parallel trial-and-error process, so the lessons learned from the past generations are not used in these types of MHs. To overcome some of these inefficiencies, a new class of EAs has been proposed as Learnable Evolution Models (LEMs) [Mic00]. In opposition to Darwinian-type EAs that contain a Darwinian evolution mode, LEM contains a learning mode wherein ML techniques are employed to generate new populations. In the learning mode, a learning system seeks reasons (rules) by particular ML techniques (e.g., AQ18 & AQ21 rule learning, C4.5 decision tree, etc.) on why certain solutions in a population (or a collection of past populations) are superior to others in performing a designated class of tasks.

Specifically, the learning mode of LEM consists of two processes [Mic00; WWH11]:

- **Hypothesis generation** – It determines a set of hypotheses that characterizes the differences between high-fitness and low-fitness solutions in recent or previous populations.
- **Hypothesis instantiation** – It generates new solutions on the basis of the learned hypotheses obtained in the hypothesis generation process. The learning mode thus produces new solutions not through semi-random Darwinian-type operations, but through a deliberate reasoning process involving the generation and instantiation of hypotheses about populations of solutions. The new populations are normally generated by injecting the extracted rules (i.e., *rule injection*) into the new solutions.

For the *hypothesis generation* process, two groups (sets) of individuals are selected from the population at each iteration: the *high performance group*, briefly *H-group*, and the *low performance group*, briefly *L-group*, based on the values of the fitness function. The collection of H-group and L-group solutions can be a subset of the population, or they can encompass the whole population [Mic00]. The *H-group* and *L-group* can be formed using two methods [Mic00]:

- **Fitness-based formation (FBF)** - In this method, the population is partitioned according to two fitness thresholds, *high fitness threshold* and *low fitness threshold*. These thresholds are presented as a percentage and determine the high and low portions of the total fitness value range in the population. These portions are then used to form the H-group and L-group of solutions. Indeed, the solutions whose fitness value is not worse than the high fitness threshold% of the best fitness value in the population form H-group, and those whose fitness value is better than the low fitness threshold% of the worst fitness value in the population form L-group. When using *fitness-based formation* method, the size of the H-group and L-group will vary from population to population, since it depends on the range of solutions' fitness values at each iteration.
- **Population-based formation (PBF)** - In this method, the H-group and L-group are formed according to two thresholds expressed as the percentage of the number of solutions in the population. These thresholds are called *high population threshold* and *low population threshold*. The high population threshold% of the best

solutions form H-group and the low population threshold% of the worst solutions form L-group.

The above two methods can be applied to the entire population as a *global* approach, or they can be applied to different subsets of the population as a *local* approach. The idea behind a local approach is that different solutions of the population may carry different information, and there would be no global information that can characterize the whole population.

Once H-group and L-group are formed, a ML technique is employed to generate qualitative descriptions that discriminate between these two groups [Mic00]. The description of an H-group represents a hypothesis that the landscape covered by H-group contains the solutions with higher fitness values compared to the landscape of L-group. Therefore, the H-group's qualitative description can be interpreted as the search direction toward the promising areas. LEMs account for such qualitative descriptions to guide the evolution process, rather than relying on semi-blind Darwinian-type evolutionary operators. Such an intelligent evolution in LEMs leads to the detection of the right directions for evolution; hence, making large improvements in the individuals' fitness values.

Two versions of LEM are introduced in the literature [Mic00]: the *uniLEM* and *duoLEM*. In *uniLEM* version, the evolution process is solely conducted through the learning mode, while in *duoLEM* version both Darwinian and learning evolution processes are coupled.

3.5.2.1 Literature classification & analysis

This section classifies and analyzes the relevant papers wherein LEM has been used to solve COPs. Table 9 classifies the relevant studies based on different characteristics related to the design and implementation of LEM including *hypothesis generation*, *hypothesis instantiation*, *group formation*, *uniLEM/duoLEM* evolution version, the *MH* algorithm and the *COP* under study.

A set of insights can be extracted from Table 9. In terms of *hypothesis generation*, the learning mode of LEM can potentially employ different learning methods that can generate descriptions discriminating between classes of individuals in a population (i.e., H-group versus L-group). If individuals are described by a vector of values (e.g., a permutation of a number of cities in TSP or a number of jobs in FSP), the rule learning methods such as AQ learners [KM00; Dom+04; WWH11; WWH12; Mor19], decision tree learners such as C4.5 [Jou+05; JKK18], or ANN can be utilized. It can be stated from Table 9 that AQ learners, in which the learning systems employ some form of AQ algorithms, are particularly suitable for implementing LEM.

Regarding the *hypothesis instantiation*, it can be seen that the majority of the studies have used a *rule injection* mechanism, wherein H-group and L-group of individuals are investigated and their strengths and weaknesses are described in terms of rules. Taking TSP as an example, a partial sequence of cities that frequently appears in H-group individuals could be a rule, and such rules can be used to evolve the population by injecting frequent sequences to create new solutions. However, due to the complexity of the COP at hand in terms of prior knowledge on the structure of the solutions and descriptive characteristics of H-group and L-group of individuals, the learning program uses an abstract, rather than precise, specification of different individuals [Jou+05; JKK18]. Consequently, the learned rules are also referred to as an abstraction of individuals. The system is then able to instantiate these abstract rules in many ways to generate new solutions in further populations. The rule instantiation process must, however, follow the constraints of the COP at hand.

Table 9 – Classification of papers studying LEM

Ref.	Hypothesis gen.	Hypothesis inst.	Group frmt.	uni/duoLEM	MH	COP
[KM00]; [Dom+04]	AQ18	Sequence injection	FBF	duoLEM	EA	HEDP
[Jou+05]	C4.5	Rule injection	FBF	uniLEM	EA	WDSDP
[WWH11]	AQ21	Rule injection	FBF	duoLEM	GA	VRP
[WWH12]	AQ21	Rule injection	FBF	duoLEM	MA	VRP
[WT17b]	Edge-intersection	Rule injection	PBF	duoLEM	GA	TSP
[JKK18]	C4.5	Rule injection	FBF	duoLEM	HS	WDSDP
[Mor19]	AQ18	Sequence injection	FBF	uniLEM	EA	VRP

3.5.2.2 Discussion & future research directions

This section identifies the challenges of implementing LEM for solving optimization problems, particularly COPs. Throughout the discussion, future research directions are also elaborated.

Considering the group formation in Table 9, there is a requirement for implementing LEM as well as an important challenge when employing fitness-based formation to create H-group and L-group. Indeed, the fundamental assumption underlying LEM is that there is a method for evaluating the performance of individuals in evolving populations. Consequently, the ability to determine the fitness value of an individual, or an approximation of this value, is a precondition for the LEM application. Therefore, LEM cannot be implemented for COPs for which defining or even approximating the fitness function is not possible.

A challenge related to the fitness-based formation that may happen in particular COPs is that in some evolutionary processes, the fitness function may not be constant throughout the entire process and change over time. It happens for particular COPs wherein the parameters change in a piece-wise manner depending on the level of the decision variables. A change in the fitness function may be gradual (i.e., fitness function drift) or abrupt (i.e., fitness function shift) [Mic00]. Indeed, if the fitness function is changing during the evolution process, some high-fitness individuals (i.e., H-group) in a previous population may become low-fitness individuals (L-group) in a future population and vice versa.

One way to overcome this challenge is keeping a record of the L-groups determined in past populations and not only the current population. Therefore, a set of past L-groups plus L-group in the current population become the actual L-group supplied to the learning mode. The number of past L-groups to be taken into consideration is controlled by a parameter. It is worth mentioning that there is no significant need to store past H-groups, because the current H-group inherently contains the best individuals until now. Generally, the formation of H-group and L-group from the current population in the evolution process ignores the history of evolution [Mic00].

An issue that may happen when using population-based formation is the possibility of an intersection between H-group and L-group. This issue should be resolved before hypothesis generation. Simple methods can be used for handling this issue such as ignoring inconsistent solutions, including inconsistent solutions in H-group or L-group, or employing statistical methods to solve the inconsistencies [Mic00].

Regarding the way of coupling LEM and Darwinian evolution modes, there are research papers [Dom+04; WT17b] that place the learning mode before Darwinian Evolution mode. On the other hand, LEM can start with Darwinian Evolution mode [WWH12], and then the two modes can alternate until the LEM termination criterion is met.

It has been regularly mentioned that involving discriminant descriptions provided by ML techniques in EAs significantly accelerates the search process toward promising solutions.

Such accelerations have been shown as frequent quantum leaps of the fitness function that signify the discovery of the correct direction of the evolution process. However, such an evolutionary acceleration imposes a higher computational complexity on the search process. This extra complexity mostly depends on the ML technique used. Therefore, employing efficient ML techniques and a parallel implementation of the AQ algorithm can reduce the complexity. Among them, the latter can reduce the complexity from linear to logarithmic.

Although the initial results from employing LEM to solve COPs are promising, there are still plenty of unsolved questions that require further research. The first attempt for future research direction should be doing numerous systematic theoretical and experimental implementations of LEM to better understand the trade-off between LEM and Darwinian evolution modes.

Among COPs, LEM has been mostly implemented on routing (e.g., TSP, VRP) and design (e.g., HEDP, WDSDP) problems. Therefore, the second important future research direction for the interested researchers is implementing LEM on other COPs to figure out the strengths, weaknesses, and limitations of both LEM and Darwinian evolution modes and to identify the most appropriate areas for their application.

3.5.3 Neighbor generation

After selecting the most appropriate operator in Section 3.5.1, it is the time to generate the neighbors from the current solution(s) using the selected operator(s). One naive way to generate neighbors is to generate all possible neighbors and select the ones with the best OFV. Another way is to generate neighbors randomly. However, considering the computational time and the goal to lead the search process towards promising areas of the search space, both strategies might not be very efficient. To be as efficient as possible, ML techniques can be used to leverage the generation of good neighbors by extracting knowledge from the generated good solutions so far.

Indeed, using ML techniques, we can extract the common characteristics that are often present in good solutions during or before the search, and use this knowledge to generate new solutions with the same characteristics by fixing or prohibiting specific solution characteristics. In this way, we can lead the search towards promising areas of the search space and accelerate the process of finding a good solution. Usually, this knowledge is in the form of a set of rules or patterns that are discovered in good solutions [Arn+21]. This process is composed of two phases: *knowledge extraction* and *knowledge injection* [Arn+21]. In knowledge extraction, the common characteristics found in good solutions are extracted. Then, in knowledge injection, the extracted knowledge is used to generate the neighbors. When the knowledge appears as a set of patterns, the most frequent patterns of good solutions are injected into the new solutions to generate the neighbors.

The knowledge extraction phase can occur either *offline* or *online*. In offline extraction, knowledge is extracted from a set of training instances with the aim to generate good solutions for new instances. However, in online extraction, knowledge is extracted from good solutions obtained during the search process, while solving the problem instance. The most common ML techniques in neighbor generation are Apriori algorithms for ARs, RL, and DT.

3.5.3.1 Literature classification & analysis

Table 10 classifies the papers using ML techniques for neighbor generation based on different characteristics such as *learning* mechanism, the *ML technique* used to extract

knowledge, the *MH* algorithm that conducts the search process, the *operator* to generate new neighborhood, the *COP* under study, and the *size* of the training set for studies that have used *offline* knowledge extraction.

Table 10 – Classification of papers studying neighbor generation

Ref.	Learning	ML tech.	MH	Operator	COP	Size
[San+06]	Online	Apriori	GA	XRO	VRP	–
[RPM06; Bar+13]	Online	Apriori	GRASP	RRO	SPP	–
[ZHD16]	Online	RL	LS	RRO	GCP	–
[Arn+21]	Online	Apriori	GA, GLS	LSO	VRP	–
[TZ19]	Online	Apriori	VNS	LSO	SMSP	–
[Sad+19]	Online	Apriori	ABC	LSO	MAX-SAT	–
[AS19]	Offline	DT, SVM, RF	GLS	LSO	VRP	192000
[Fai+19]	Online	RL	ABC	LSO	TSP	–
[Wan+20b]	Online	RL	LS	RRO	WIDP	–
[ZHD20]	Online	Apriori	EAs	MPO	QAP	–
[AD+18]	Online	Apriori	GRASP	LSO	VRP	–

As can be seen in Table 10, almost all reviewed papers have used the online manner to extract knowledge in the form of patterns. Indeed, when dealing with a new instance with unknown characteristics, the most efficient way to extract patterns is online. An advantage of online pattern extraction is avoiding the misleading patterns extracted in an offline manner that may not correctly represent the properties of the good solutions of the new problem instance; however, the computational overhead of online pattern extraction should not be ignored.

3.5.3.2 Discussion & future research directions

In this section, the requirements and challenges of applying ML techniques for neighbor generation are discussed. Then, some directions for future research are presented.

One of the most important requirements of using ARs is *data availability*. In fact, when extracting patterns, it is indispensable to have a sufficient pool of good solutions, since the accuracy of the extracted patterns directly depends on the availability of sufficient data. The higher the availability of data, the higher the precision and usefulness of the extracted patterns.

The first challenge of neighbor generation is to determine the stage of the search process at which the knowledge should be extracted to generate new neighbors. This challenge is twofold; first, in the beginning of the search process, the improvements are larger, and the set of good solutions frequently change, and no precise pattern can be extracted from the pool of good solutions. Therefore, the knowledge should be extracted after some iterations are passed. Second, if the knowledge is injected into the neighbor generation process in the earlier stages of the search process, it prevents the MH to explore different areas of the search space, and it may cause a premature convergence to a local optimum.

The second challenge is related to the frequency at which the extracted pattern should be updated and injected to create new neighbors. Indeed, one may identify the patterns once and use them for neighbor generation throughout the search process, or update the patterns frequently based on the characteristics of the newest good solutions. Pattern extraction may be a time-consuming process which increases the computational cost of the search process. Therefore, there is a trade-off between the accuracy of patterns based on the latest information from the search process and the computational overhead of the pattern extraction process.

Another challenge arises when several pieces of patterns are available in good solutions and there might be plenty of possibilities to inject them into new solutions (i.e., separate or combined injection of patterns). There is no guarantee that the combination of pieces

of patterns also provides good solutions [Arn+21]. In other words, although several single patterns may appear in a large number of good solutions, their combination does not necessarily generate better or even good solutions. For instance, in permutation-based representations, edges $A-B$ and $C-D$ may separately appear in good solutions, however; there is no guarantee that edges $A-B$ and $C-D$ simultaneously lead to a good solution.

Along with all the previously mentioned challenges, last but not least is making a decision about the ratio by which new solutions are generated using the extracted knowledge. The level of such ratio affects the behavior of the MH in terms of its exploration and exploitation abilities. If almost all solutions are generated based on the previous knowledge, the MH tends to mostly exploit the currently observed area. On the other hand, if a small ratio of solutions is generated based on the previous knowledge, the algorithm has an opportunity to explore new areas of the search space. Therefore, the user has to consider the exploration and exploitation abilities of the algorithm while deciding about the ratio of the solutions to generate based on the extracted knowledge.

Considering Table 10, most of the studies that used ARs attempt to identify the characteristics of the good solutions. One research direction could be using ARs to identify the characteristics of bad solutions which have to be removed from the new solutions. Then, this knowledge could be used merely to avoid bad solutions, or it could be used besides the knowledge obtained from good solutions to complement the patterns of good solutions.

In addition, most of the papers in the literature have used the knowledge of good solutions to exploit the most promising areas of the search space, while the exploration aspect of the search process should be also taken into consideration. One way of doing that is extracting the rare patterns from the visited solutions and injecting them into new solutions to generate solutions far from the solutions visited so far.

3.6 Parameter setting

The success of any MH significantly depends on the values of its parameters [Tal09]. As the parameters control the behavior of the algorithm during the search process, the values of parameters should be properly set to obtain the highest performance. Although there are several suggestions on the values of parameters for a similar group of problem instances in the literature, they are not necessarily the most appropriate settings when solving the problem instances at hand [WM97]. Indeed, parameter setting is not a onetime task, and researchers need to set the algorithm's parameters whenever they solve new problem instances [HLY19]. Parameter setting, also known as *algorithm configuration* [Hoo11], is divided into two categories, *parameter tuning* and *parameter control* [EHM99].

- **Parameter tuning** – Also known as *offline* parameter setting, identifies appropriate parameter levels before employing the algorithm to solve the problem instances at hand. In this case, the levels of parameters remain unchanged during the execution of the algorithm. Parameter tuning can be done using different methods such as *brute force* experiments [Pha+19], *Design of Experiments* (DOE) [Tal09], *racing* procedures [HLY19], and *meta-optimization* [Tal09].
- **Parameter control** – Also known as *online* parameter setting, focuses on adjusting the levels of parameters of an algorithm during its execution, rather than using initially fine-tuned parameters that remain unchanged during the whole execution. Parameter control methods have been developed based on the observation

that tuning the parameters does not necessarily guarantee the optimal performance of a MH, since different settings of a parameter may be appropriate at different stages of the search process [Ale12]. The reason is attributed to the non-stationary search space of optimization problems that results in a dynamic behavior of the MHs, which should evolve regularly from a global search mode, requiring parameter values suited for the exploration of the search space, to a local search mode, requiring parameter values suitable for exploiting the neighborhood. Parameter control can be performed in three manners [KHE14]; *deterministic* manner in which the levels of parameters are adjusted using given schedules (e.g., pre-defined iterations) without no feedback from the search process, *adaptive* manner in which the levels of parameters are adjusted using feedback from the search process, where a credit is assigned to the parameters levels based on their performance, and *self-adaptive* manner in which the parameters levels are encoded into solution chromosomes and evolved during the search process.

ML techniques can be employed in both parameter tuning and parameter control. In parameter tuning, ML techniques such as LogR [Ram+05], LR [CR09], SVM [LCA11], and ANN [Dob10] are used to predict the performance of a given set of parameters based on a set of training instances. In parameter control, ML techniques can involve in *adaptive parameter control* to help control the parameters levels by using feedback information on the performance of the parameters levels during the search process. The integration of ML techniques in adaptive parameter control is similar to that of AOS (Section 3.5.1), where feedback is used to adapt the parameters levels to the search space. It similarly involves four main steps (except Move Acceptance step): 1) performance criteria identification, 2) reward computation, 3) credit assignment, and 4) selection, which have been explained in detail in Section 3.5.1.

3.6.1 Literature classification & analysis

This section aims at classifying, reviewing, and analyzing the studies on the use of ML techniques in the parameter setting of the MHs. In this regard, Table 11 classifies the papers based on different characteristics including parameter *tuning/control*, the employed *ML technique*, *credit assignment* and *selection* methods, the *MH* for which the parameters are set, the *parameters* to set, the *COP* under study, and finally the *size* of the training set for the papers that have studied *parameter tuning*.

Table 11 – Classification of papers studying parameter setting

Ref.	Tuning/ control	ML tech.	Credit ast.	Selection	MH	Parameter	COP	Size
[Hon+02]	Control	RL	SCA	MCS	GA	Crossover & mutation rates	KP	–
[Ram+05]	Tuning	LogR	–	–	EA	Population size	TSP	25
[MR07]	Control	RL	SCA	MCS	GA	Crossover & mutation rates	JSP	–
[CR09]	Tuning	LR	–	–	GH	Population size	CLSP	4992
[ZEC10]	Tuning	SVM	–	–	TS	Intensification rate	TSP, VRP	25000
[LCA11]	Tuning	SVM, LR	–	–	PSO	Learning rates	WDSDP	5284
[LT12]	Control	RL	SCA	MCS	EA	Mutation rate	PDP	–
[Ale+14]	Control	LR	–	–	EA	Crossover & mutation rates	QAP	–
[AP14]	Control	RL	SCA	PMS	DE	Perturbation & Mutation rates	KP	–
[Seg+16]	Control	RL	SCA	MCS, PMS	MA	Mutation rate	AP	–
[BEB17]	Control	RL	QLCA	SMS	BLS	Number & probability of perturbation	VSP	–
[Che+20]	Control	RL	QLCA	EGS	GA	Crossover & mutation rates	FSP	–
[Özt+20]	Control	RL	QLCA	EGS	VNS	Acceptance & QL parameters	FSP	–

As can be seen from Table 11, parameter control has attracted much more attention compared to parameter tuning when solving COPs. Indeed, the main reason is that fixed values of the parameters do not necessarily guarantee the best performance of a

MH during the whole search process. The underlying cause is the non-stationarity of the search space, as explained at the beginning of this section. From a general point of view, the performance of a MH significantly depends on its capability to explore and exploit the promising areas of the search space, and the exploration and exploitation abilities of a MH depend on the levels of its parameters. Taking the GA as an example, the crossover and mutation rates should change depending on the performance of the algorithm and the properties of the search space. For example, once a promising solution is explored, that solution should be exploited carefully. Therefore, the mutation and crossover rates should be decreased and increased, respectively. On the other hand, when the algorithm gets stuck in local optima, the mutation rate can be increased to help the solution to escape from the local optima.

Based on Table 11, ML techniques based on RL have been mostly employed when controlling the parameters during the search process. The underlying reason is that RL agent iteratively learns from interactions with its environment to take actions that would maximize the reward. In the context of parameter setting, a list of different configurations can be defined as a set of actions and each time a configuration set provides better solutions, a reward is assigned to that set of configurations. Indeed, at the beginning of the search process, all possible configuration sets have the same probability to be selected. During the search process, these probabilities can change according to their success in creating better solutions [AP14]. It has been reported in the literature that in the earlier stages of the search process, the selection probability of each configuration set changes more often compared to the latter stages of the search process. It has been explained by the diversity loss that occurs during the search process [LT12; AP14; Seg+16; BEB17]. In addition, there is evidence that the levels of exploration representative parameters (e.g., mutation rate in GA and DE, size of Tabu list in TS) change more frequently at the earlier stages of the search process when the MH is exploring the search space. On the other hand, the exploitation representative parameters get more attention in the latter stages of the search process, when the MH needs to exploit promising solutions found so far [ZEC10; LT12; AP14; BEB17].

Another insight from Table 11 is that most of the papers have done parameter setting for population-based MHs. Population-based MHs possess both exploration and exploitation representative parameters, and a balance between these abilities should be well established. If not, the search process either gets stuck in local optima or performs a random search.

3.6.2 Discussion & future research directions

The first challenge when performing parameter setting is to decide whether to tune or control the parameters. Each mode of setting has its own advantages/disadvantages. There are experimental evidence revealing that the optimal parameter settings are different not only for different problem instances, but also for different stages of the search process of the same problem instance. In this situation, it is recommended to perform parameter control despite the computational overhead imposed on the search process. A big challenge related to the parameter control is the trade-off between exploration and exploitation to select the current best configuration(s) or search for new good ones. Once the configuration is changed during the search process, the new configuration should work for a certain number of iterations so that its performance can be evaluated. An extra parameter should be then defined to control when the configuration should be changed (i.e., the rate of configuration change). The rate of configuration change itself is another parameter that needs to be tuned or controlled during the search process. Therefore, the parameter control itself introduces a set of other parameters (i.e., parameters of the

parameter control mechanism) that need to be tuned or controlled, which results in the increase of the complexity of the parameter control.

Parameter control faces another challenge when dealing with continuous parameters, where an infinite number of values exist for each parameter. One way to deal with this challenge is considering the parameter setting as a separate optimization problem and the parameter levels as decision variables to be optimized. The other way is developing a self-adaptive parameter control mechanism. Another way studied in the literature is subdividing the levels of parameters into feasible intervals [Ale12]. In this way, the interval borders are normally fixed and not manipulated by the search process. As a result, the number and the size of the intervals have to be determined by the user a priori that may jeopardize the accuracy of the interval as well as the efficiency of the MHs. If the levels of a parameter are divided into several narrow intervals, the accuracy of the intervals would be higher, while the computational effort of selecting among the intervals significantly increases. Accordingly, there is a risk to find good intervals late, or there might be some intervals that are not selected at all. If the intervals are wide, different parameter values belonging to a single interval may lead to different performance of the MH, as wide intervals may encompass smaller intervals that behave differently. As a result, no unique performance behavior can be attributed to such wide intervals. To cope with this issue, adaptive range parameter control method [AMM12] adapts intervals during the search process, and entropy-based adaptive range parameter control method [AM13] clusters parameter values based on their performance. However, more research is indeed required to understand the impact of small changes in the values of continuous parameters on the performance of the MHs.

Considering Table 11, one future research direction is applying parameter setting methods to other MHs such as single-solution based MHs. A first try can be developing a parameter setting mechanism to control the parameters of SA, such as the cooling temperature, which is always a challenging problem for practitioners. Additionally, further investigation could be done on controlling parameters that have received little attention so far.

Another research direction that should be put in priority is implementing parameter setting on multi-objective COPs using ML techniques. It should be noted that one of the challenges of adaptive parameter control in multi-objective optimization problems is how to define the feedback with a single value such that it would be representative of the quality of a parameter value over multiple objective functions.

3.7 Cooperation

Different MHs with particular strengths and weaknesses working on the same problem instance produce different results. In this situation, using a framework enabling the use of different MHs in a cooperative way could result in an improved search process. The main motivation of developing cooperative MHs is to take advantage of the strengths of different MHs in one framework, to balance exploration and exploitation, and to direct the search towards promising regions of the search space [Mar+11]. The interest in using such frameworks for solving COPs has risen due to their successful results [Tal02; Mar+16]. The cooperation framework can be modeled as a multi-agent system in which the search process is performed by a set of agents that exchange information about states, models, entire sub-problems, solutions, or other search space characteristics [BR03].

In multi-agent based cooperative MHs, each agent could be a MH or a MH's component such as an operator, a search strategy, a solution, etc. that tries to solve the problem,

while communicating with other agents [Sil+18]. The cooperation could happen either at the *algorithm level* (between several MHs) wherein different MHs with specific characteristics cooperate to solve a COP, or it can happen at the *operator level* (inside a MH), wherein different operators cooperate when discovering different regions of the search space. The former belongs to the category of high-level integration of ML techniques into MHs while the latter falls in the low-level category of integration.

ML techniques can help in designing intelligent cooperation frameworks by extracting the knowledge from the resolution of the problem instances by different agents (MHs). This knowledge is then incorporated into the framework that enables the framework to continuously adapt itself to the search space. In this way, ML techniques can improve the overall performance of the cooperation framework. The integration of ML techniques into a cooperation framework can happen in two learning levels: *inter-agent* and *intra-agent* levels. The former is to adapt the behavior of the overall framework to the search space, while the latter is to adapt the behavior of each agent to the search space.

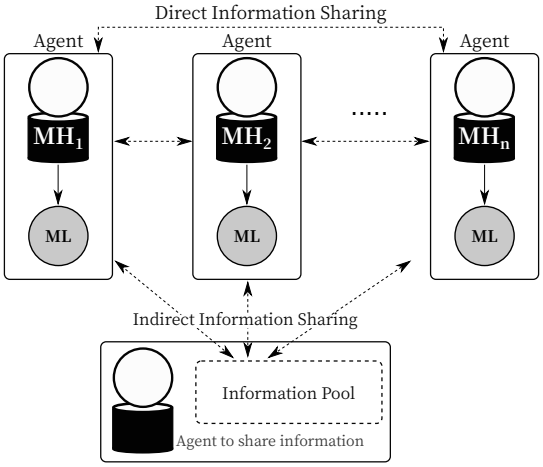

Figure 9 – Cooperation procedure

Figure 9 illustrates the process of cooperation between agents. Agents can cooperate *sequentially* or in *parallel* [Tal02]. The cooperation between the agents can be *synchronous*, where the agents work in a parallel way and none of them waits for the results from other agents, or *asynchronous*, otherwise [Mar+16]. As can be seen in Figure 9, the exchange of information between the agents is *direct* (many-to-many), where each agent is allowed to communicate with any other agent and *indirect*, where agents are only allowed to use the information provided in a common pool [Mar+16]. While cooperating, the agents can share *partial* or *complete* solutions to proceed the search process.

3.7.1 Literature classification & analysis

Table 12 classifies the papers studying cooperation for COPs based on different characteristics such as *cooperation level*, *parallel/ sequential* mode of cooperation, *learning level*, the employed *ML technique*, *direct/ indirect* information sharing, *solution sharing* type between the agents, the *MH* algorithms participated in the cooperation, and finally the *COP* under study. To the best of our knowledge, Table 12 reviews the most relevant papers, including the most recent papers in the literature that study the cooperation between MHs (or MHs' components) to solve COPs using ML techniques.

Table 12 – Classification of papers studying cooperation

Ref.	Coop. level	Parl./ Seq.	Learning	ML tech.	(In)Direct	Sharing	MH	COP
[LBCK05]	Alg.	Parl.	Inter	Apriori	Ind.	Part.	TS	VRP
[MCK08; MCK09; MKC10]	Alg.	Parl.	Intra	RL	Di.	Comp.	EA	VRP, FLP
[CGM09]	Alg.	Parl.	Inter	DT	Ind.	Comp.	GA, SA, TS	KP
[Bar10a]	Opr.	Parl.	Inter	RL	Ind.	Comp.	LS	VRP
[Sil+15]	Alg.	Parl.	Intra	LA	Ind.	Comp.	ILS	VRP
[LA16]	Alg.	Parl.	Inter	LA	Ind.	Comp.	GA, DE, SA, ACO, GD, TS	MSP
[Mar+16]	Alg.	Parl.	Inter	Apriori	Di.	Part.	MHs	FSP, VRP
[Sgh+15; SJG18]	Opr.	Seq.	Inter	RL	Ind.	Comp.	GA, LS	QAP, KP, GCP, WDP
[WT17a]	Alg.	Seq.	Inter	k-means	Di.	Comp.	MA, LS	FSP
[Sil+19]	Alg.	Parl.	Intra	QL	Ind.	Comp.	ILS	VRP, PMSP
[KM+20a]	Alg.	Seq.	Inter	k-means	Di.	Comp.	DE, ILS	IPP
[KM+20b]	Alg.	Seq.	Inter	k-means	Di.	Comp.	GA, ILS	HLP

As can be seen in Table 12, the majority of the studies applied cooperation in a *parallel* manner. The main motivation has been an attempt to reduce the computational time of executing several MHs one after another (i.e., sequential cooperation). In Parallelism, the MHs are executed simultaneously, and consequently, it results in the reduction of the search process time. Indeed, the combination of cooperation and parallelism allows self-sufficient algorithms to run simultaneously while exchanging information about the search [Sil+18]. This combination has attracted increasing attention in optimization, especially for solving COPs, since they have shown good results on different COPs [Mar+16; KM+20a]. Moreover, as mentioned by [Tal02] and [CTA05], the best results obtained for many optimization problems are achieved by the cooperative algorithms. In addition, there is greater access to parallel computing resources, which provides new possibilities for developing these techniques [Sil+18].

Regarding *synchronous* and *asynchronous* cooperation, the majority of the studies focus on the asynchronous way of cooperation. Indeed, in most of the time when MHs have been executed in parallel, their cooperation has been asynchronous, wherein none of MHs wait for the results of the others. Asynchronous cooperation carries fewer operational challenges and gives the opportunity to modify the cooperation framework easily. Using asynchronous cooperation, MHs can be added or dropped easily without any change to the overall framework. On the other hand, a synchronous cooperation faces more challenges. In synchronous cooperation, the agents must coordinate their actions in time. In other words, the agents are activated only when all agents are ready to act [Bar10b]. Indeed, although the agents work independently, the activation times of the agents depend on each other. Accordingly, there is a need to determine a synchronization point in which the agents announce their readiness and start to act. Hence, the time dependence of the agents may cause some agents to wait, and consequently some processors stay idle for a period of time.

According to Table 12, the agents of a cooperative system could be the MHs or MHs' components. The cooperation could happen either at an *algorithm level* or *operator level*. The difference between the cooperation at the operator level and AOS (Section 3.5.1) relies on the fact that in AOS, operators are selected one after another based on their history of performance, while in the cooperation framework, the operators share information while searching for solutions cooperatively.

The main ML techniques used in cooperative MHs are RL and Apriori algorithms for ARs. RL has been used to help the system adapt its behavior based on the experience it gains throughout the search process. RL is used in two levels; within the agents (intra-agent level) to adapt their behavior to the characteristics of the search space during the search process by modifying their components (selecting the operators) and in a higher

level (inter-agent level) to adapt the application of the agents based on their performance compared to other agents. On the other hand, ARs are used to identify the common characteristics of good solutions. Then, this knowledge is shared among the agents in the form of partial solutions, which allows each agent to generate new solutions based on the identified patterns and guide the search toward promising regions.

3.7.2 Discussion & future research directions

This section aims at introducing the requirements and potential challenges when designing a cooperation framework of MHs. Next, a set of future research directions are provided.

The design and implementation of efficient cooperative MHs require sufficient apriori knowledge about different MHs. To take advantage of the strengths of different algorithms, which is the main motivation of developing cooperative algorithms, one needs to be aware of a broad spectrum of algorithms and have knowledge on their strengths and weaknesses. For instance, population-based MHs are powerful in exploration. On the other hand, single-solution based MHs are strong in exploitation. As can be seen in Table 12, studies with heterogeneous algorithms have incorporated both population-based and single-solution based MHs into their cooperation framework to take advantage of both exploration and exploitation abilities. As discussed earlier, the information between the agents can be shared in the form of partial solutions, where ARs can be used to generate these partial solutions. In this regard, a set of challenges in front of ARs for partial solution generation, which were elaborated in Section 3.5.3, also needs to be addressed in the design of cooperation frameworks.

Considering Table 12, in most of the studies, the agents (MHs) attempt to save and share good obtained solutions partially or completely. In this way, each MH would be aware of the promising regions exploited by other MHs. As a future research direction, sharing the bad solutions and their corresponding characteristics could be also useful to prevent MHs to explore non-promising regions. Indeed, the non-promising regions already visited by a MH could be prohibited to be explored and exploited again by other MHs.

Most of the reviewed papers in this section have used cooperative MHs to solve single-objective COPs, and there are only few papers that study cooperation in multi-objective COPs [KM+20b; KM+20a]. These two papers have used k-means to link multi-objective population-based MHs with single-solution based MHs. Once the non-dominated solutions are obtained via the population-based MHs, k-means is used to cluster these solutions. Then, the representative of each cluster is given to the single-solution based MH to be more exploited. This cooperation has led to better non-dominated solutions in terms of both the quality and the computational time of the search process. In this regard, another future research direction could be extending the concept of cooperation to multi-objective COPs.

3.8 Conclusion

In recent years, ML techniques have been extensively integrated into MHs for solving COPs, and promising results have been obtained in terms of solution quality, convergence rate, and robustness. In this chapter we provided a comprehensive and technical review on the integration of ML techniques in the design of different elements of MHs for different purposes including algorithm selection, fitness evaluation, initialization, evolution,

parameter setting, and cooperation. Throughout this chapter, a set of requirements, challenges, and insights have been elaborated for each purpose of integrating.

The prominent difference of this chapter compared to the existing review studies in the literature is its technical overview on each way of ML-into-MH integration. This comprehensive and technical survey provided a complete description of all the key concepts and preliminaries (on MHs, ML techniques, and COPs) and then, proposes a taxonomy to provide a common terminology and classification, followed by a technical discussion on the advantages/limitations, requirements, and challenges of implementing each way of integration, and finally, terminated with providing promising future insights and directions.

We observed that integrating ML techniques into MHs mostly leads to an overall improvement of the MH's performance; however, each type of integration carries its own challenges and limitations. The challenges mostly relate to the extra complexity that ML techniques introduce to the search process, particularly the learning process of ML techniques. This complexity is often acceptable when significant improvement is obtained in the MH's performance. However, is the performance improvement is not that significant, the integration of ML techniques into MHs might be questioned (unnecessary). There is also a common challenge beyond the particular way of ML-into-MH integration; that is, introducing new parameters (i.e., ML's parameters) to the algorithm that need to be well tuned to guarantee the best performance of ML techniques. This tuning should be done very carefully, and it might be time-consuming and even risky.

Another overall conclusion of this survey study was that the majority of studies in the literature attempt to use simple but efficient ML techniques. This intention is, in one way, to limit the extra complexity that ML techniques introduce to MHs, and also to provide the highest performance improvement for the MHs. Accordingly, although complex and advanced ML may provide more concrete knowledge, they do not necessarily yield better overall performance, particularly when the computational time is limited.

Now, considering the terminology and classification we provided in this chapter, we focus on *operator selection*, a subclass of *evolution*, in Chapter 4. We propose a general framework to integrate ML into MHs to select appropriate search operator.

Chapter 4

Q-learning for Operator Selection: A General Framework

Contents

4.1 General framework 86
4.2 Application to TSP 89
4.2.1 Set of search operators 89
4.2.2 QILS framework to select local search operators 93
4.2.3 QILS framework to select perturbation operators 95
4.3 Application to PFSP 95
4.3.1 Set of search operators 96
4.3.2 QILS framework to select perturbation operators 97
4.4 Experimental design 99
4.4.1 Dataset 99
4.4.2 Benchmark comparison 99
4.4.3 Performance comparison metrics 101
4.4.4 Statistical test 101
4.4.5 Parameters tuning 102
4.5 Computational results 104
4.5.1 Application of QILS to TSP 104
4.5.2 Application of the QILS framework to PFSP 107
4.5.3 Complexity analysis 129

In Chapter 3, we showed that ML can be integrated into MHs for different purposes including *algorithm selection*, *fitness evaluation*, *initialization*, *evolution*, *parameter setting*, and *cooperation*. In this chapter, we focus on *operator selection*, a subclass of *evolution* and propose a general framework on how ML can be integrated into MHs to learn to select appropriate search operators during the search process. This is the second contribution of this thesis that has been published in the *European Journal of Operational Research* [KM+22].

We, first, propose a general framework that is able to automatically select the search operators without the need for expert knowledge. Then, we apply the framework to solve two COPs, TSP and PFSP. In order to apply the framework to solve these problems, first we need to determine the properties of Q-leaning based on the goal for which we use the Q-learning, and to introduce the problem-specific information such as the set of available operators for these problems. Next, we design a set of experiments explaining the employed datasets, parameter setting procedure, and the performance metrics by which the framework has been evaluated. Finally, the obtained numerical results have been provided for both applications and the complexity of the framework has been investigated.

4.1 General framework

There is a rise in the number and variety of problem-specific operators for efficiently solving optimization problems. Selecting and applying these operators within a MH requires much expertise in the domain. That is especially the case for COPs with plenty of proposed problem-specific operators, where the classical operators are not as competent as problem-specific ones. This issue highlights the necessity of an automatic approach to select the most appropriate operator(s) based on their performance without having an expertise in the domain. In this way, even inexperienced users are able to select appropriate operators for solving COPs.

In this section, we aim to propose a framework to show how ML can be integrated into MHs to automatically select the search operators without injecting experts knowledge into the selection process. Sometimes, the expert knowledge may not be sufficient/optimal and may lead to decisions with unsatisfactory results. Consider a MH with a set of multiple search operators. The goal of operator selection is to decide which operator to be selected and applied at each step of the search. The quality (performance) of the operators at each step of the search depends on two criteria: first, the stochastic nature of the evolution process where some seemingly poorly performing operators might just have been unlucky at some steps and second, the region of the search space being explored and its landscape characteristics where a good operator might show a poor performance in some regions. Therefore, it would be hard to forecast the performance of the operators at each step of the search a priori and to determine the best operator using the expert knowledge, since this performance does depend on the landscape characteristics of the problem instance at hand as well as the stochastic nature of the evolution. Therefore, the expert knowledge may not give us the optimal operator and consequently the optimal decision.

To deal with this issue, we propose to substitute expert knowledge with ML techniques to automatically select the operators. Among different ML techniques (i.e., supervised learning, unsupervised learning, semi-supervised learning, and RL), we propose to use RL. In supervised learning, the expert must provide the set of input as well as the expected output (behavior) to train the machine learning model and the goal is to blindly mimic the expert and not to optimize some performance measure. However,

sometimes the expert knowledge is not optimal/satisfactory. In this condition, RL can come into play to train a model through trial and error without the need for expert knowledge. This model is able to explore the space of possible decisions and learn out of the experience the best performing behavior. Furthermore, in supervised learning, the learned behavior may not generalize well to unseen instances. On the contrary, in RL, for a given reward, the model learns to find the best behavior for each single instance.

Figure 10 shows the mechanism of the proposed framework. Furthermore, the detailed procedure of the QILS framework has been described in Algorithm 2. According to Figure 10, the proposed framework has two main components; a ML technique and a MH. As the ML component, we have used Q-learning, a technique based on RL. Regarding the MH, we have used the ILS algorithm (please see Section 2.2 of Chapter 2), which is a simple single-solution based meta-heuristic powerful in both exploitation and exploration. Considering the Q-learning as the ML technique and ILS as the MH, we call the proposed framework QILS hereafter.

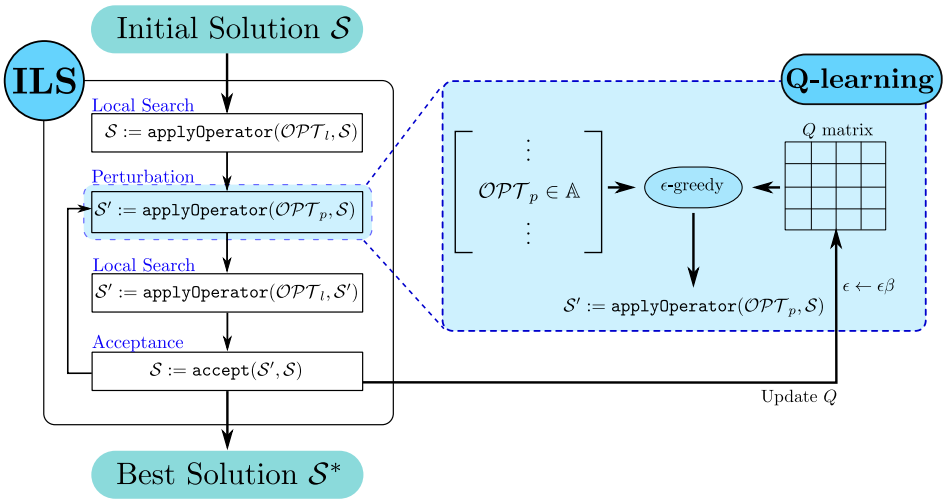

Figure 10 – Flowchart of the proposed QILS framework

The proposed QILS framework starts with the MH component, where an initial solution is first generated. The solution then goes through three main steps of ILS: the perturbation (exploration) step, the local search (exploitation) step, and the acceptance. A RL technique has been integrated into the ILS to select the most appropriate operator among a set of multiple available operators to be applied to the solution. Accordingly, depending on the user's goal, RL can be integrated into the perturbation step, the local search steps, or both. For simplicity, in Figure 10, we show a case where Q-learning, as a RL technique, has been used to select the perturbation operators in the perturbation step. In the following, we go into more details of the operator selection process.

In our selection process, we use the idea of AOS, wherein the operators are selected adaptively throughout the search process based on their quality. In addition, we take into account the concept of states in RL in our selection process. The operators are selected based on two criteria at each decision point: the performance history of the operators from the beginning of the search that represents the quality of the operator, and the current state (status) of the search that characterizes the environment under study. We model the operator selection using the Q-learning algorithm. The Q-learning is composed of three main elements: the set of states $s \in S$ that represents the status of the environment, the set of actions A that represents the set of decisions that need to

be learned though interaction with the environment, and finally, a reward function that aims to guide the model to make decisions that maximize the cumulative reward.

Algorithm 2. Pseudo code of the proposed framework

Input: $\epsilon, \beta, \alpha, \gamma, E, \eta$ // Set of parameters
Input: $\mathbb{A}$ // Set of possible actions at each state
Input: $\mathbb{S}$ // Set of states
Input: $\mathcal{OPT}_l$ // A given local search operator
Output: $\mathcal{S}^*$ // Best solution found
Function QILS($\epsilon, \beta, \alpha, \gamma, E, \eta, \mathbb{A}, \mathbb{S}, \mathcal{OPT}_l$):
  $\mathcal{S} :=$ initialSolution() // Initialize solution using a heuristic
  $\mathcal{S} :=$ applyOperator($\mathcal{OPT}_l, \mathcal{S}$) // Apply local search to the initial solution
  $\mathcal{S}^* := \mathcal{S}$ // Remember the best solution found
  $Q := [0]$ // Initialize Q, a zero-filled $|S| \times |A|$ table
  $s :=$ initialState($\mathbb{S}$) // Initialize the state
  $\mathcal{OPT}_p :=$ randomChoice($\mathbb{A}$) // Initial action is drawn randomly from A
  while ! terminationCriterion() **do**
    // Start of an episode
    $F_b := f(\mathcal{S})$ // Remember OFV of current local optimum before an episode
    $F_b^* := f(\mathcal{S}^*)$ // Remember OFV of best solution found before an episode
    $F := F_b$ // Remember OFV of the best local optimum during an episode
    $F^* := F_b^*$ // Remember OFV of the best solution found during an episode
    for $e = 1 : E$ **do**
      // Perturbation:
      $\mathcal{S}' :=$ applyOperator($\mathcal{OPT}_p, \mathcal{S}$) // Apply selected operator
      // Local search
      $\mathcal{S}' :=$ applyOperator($\mathcal{OPT}_l, \mathcal{S}'$) // Apply local search operator
      // Acceptance
      if accept($\mathcal{S}', \mathcal{S}$) **then**
        $\mathcal{S} := \mathcal{S}'$
        $F := \min(F, f(\mathcal{S}))$ // Track OFV of best local optimum during an episode
      end
      if better($\mathcal{S}', \mathcal{S}^*$) **then** // Update best solution found
        $\mathcal{S}^* := \mathcal{S}'$
        $F^* := f(\mathcal{S}^*)$
      end
    end
    // Update Q, change state s and select LLH_p (action) for next episode
    $Q, s, \mathcal{OPT}_p :=$ Q-learning($F_b, F_b^*, F, F^*, \epsilon, \alpha, \gamma, \beta, \eta, \mathbb{A}, s, LLH_p$)
  end
  return Π^*

In the QILS framework, the actions (decisions) are the operators (perturbation, local search) that need to be selected by Q-learning. On the other hand, the set of states and the reward function are problem-specific properties that are determined based on the context and the goal of using Q-learning.

In the QILS framework, the application of any action is followed by an immediate re-

ward from the environment (i.e., the problem) that shows the immediate impact of the applied operator on the search process. Next, based on this immediate reward and the operators' performance history, the Q-learning algorithm assigns a credit to each operator. Depending on the current state of the search as well as the credit of each operator, the next perturbation operator is selected to be applied. For the selection, we use the ϵ-greedy strategy to make a balance between exploiting the operators that have performed well so far and exploring other operators to give them a chance to be selected. We introduce the concept of episode in the proposed framework, wherein each action is given a chance of one episode equal to a fixed number of iterations before evaluating its performance.

The proposed framework is most useful when dealing with several competitive operators for solving a COP such that none of them can be preferred over the others a priori, and choosing the best operator exhaustively is computationally expensive. In this condition, Q-learning can be used to automatically select the operators.

It is worth mentioning that the proposed QILS framework is general and can be integrated into any MH, ranging from single-solution based to population-based MHs. Also, it can be applied to solve any COP. In order to integrate the framework into any MH and apply it to other COPs, there are two types of application-specific properties that needs to be determined: first, the properties of Q-learning including a set of states, actions and reward function that depends on the goal of using Q-learning, and second, a set of search operators that depends on the COP at hand. It would be good to consider a pool of operators with different characteristics to be able to take advantage of these characteristic simultaneously.

4.2 Application to TSP

In this section, as the first application, we apply the QILS framework to solve the traveling salesman problem. The QILS framework is applied separately for two different purposes: first, to select the local search operators of ILS with the hope to improve the exploitation ability and second, to select the perturbation operators of ILS aiming at improving the exploration ability of ILS. The set of problem-specific properties of the QILS framework including the set of search operators (local search and perturbation) for TSP, and the properties of Q-learning such as the set of actions, states, and the reward function are defined for each of these two purposes.

4.2.1 Set of search operators

Various local search and perturbation operators exist for TSP in the literature [JRR95]. Among different local search operators, we use the *basic 2-opt* operator [Tal09], and propose two new operators based on the *basic 2-opt* and *basic insertion* operators. Accordingly, we include three operators in the set of available operators as the *basic 2-opt*, an extended version of 2-opt which we call it *best-independent-moves 2-opt*, and an extended version of insertion operator that we call it *4-move insertion*. It is worth mentioning that in the QILS framework, the local search operators perform a descent-based search and continue until no more improvements are found. Considering that a solution to TSP is a Hamiltonian tour $\Gamma := (\gamma_1, \ldots, \gamma_v) \in \mathfrak{S}(V)$ of the cities, where $\gamma_j \in V$ denotes the index of the city at position j in the tour Γ, the mechanism of these operators are explained in the following:

- ***Basic 2-opt***: This operator has been motivated by the an observation in Euclidean problems [JRR95]. If a Hamiltonian cycle crosses itself it can be easily

shortened by removing the crossing edges and reconnecting the two resulting sub-routes by new edges without a cross. This makes the new route shorter than the initial one. Accordingly, the basic 2-opt operator takes two edges from the route of the cities, removes them, and reconnects the two resulting sub-routes with new edges. If the move leads to a shorter travel distance the current route will be updated [Cro58]. The pseudo code of the *basic 2-opt* local search operator is provided as Algorithm 3.

Algorithm 3. Pseudo code of the *basic 2-opt* swap local search operator

```
Input: G                                          // Weighted graph G = (V, E)
Input: D                                          // Distance matrix
Input: Γ                                          // A given tour
Input: f(Γ)                                       // Objective function value of tour Γ
Output: Γ                                         // A local optimum
Function 2opt-swap(Γ, D):
    while ! terminationCriterion() do
        for i = 1 : |V| − 2 do
            for j = i + 2 : |V| do
                δ := d_{γ_i,γ_j} + d_{γ_i+1,γ_j+1} − d_{γ_i,γ_i+1} − d_{γ_j,γ_j+1}
                if δ < 0 then                     // Update solution during the local search
                    Γ' := swap(i, j, Γ)           // Swap the permutation between cities γ_i
                     and γ_j
                    f(Γ) := f(Γ) + δ
                    break
                end
            end
        end
    end
    return Γ
```

- ***Best-independent-moves 2-opt***: We propose an extended version of 2-opt operator based on the idea of best-move 2-opt presented in [EKEB+18]. In the best-move 2-opt, in each iteration of the local search, all the improving moves are identified and sorted based on their improvement value, and only the best improving move is performed. In this way, the information gathered about other improving moves is neglected and remain unused. However, in the proposed 2-opt, the main idea is to use the gathered information about the improving moves and to perform all possible moves simultaneously as long as they can be done independently (i.e., they do not share any segment of the route). In this way, in an iteration of the local search, a greater value of improvement achieves. We call this new 2opt, the best-independent-moves 2-opt. To explain the procedure of the proposed 2-opt, consider a simple example of Figure 11. In the first step, all improving moves are identified (moves I, II, III, and IV with improving values in parenthesis). Then, the improving moves are sorted based on their improvement values in a descending order (moves II, VI, I, III). Finally, starting from the first move, all the independent moves are performed simultaneously (moves II, VI, and III). Indeed, move I cannot be applied immediately after move VI since they share the same segment "$\gamma_{18} - \gamma_{19} - \gamma_1$". The pseudo code of the *best-independent-moves 2-opt* local search operator is provided as Algorithm 4.

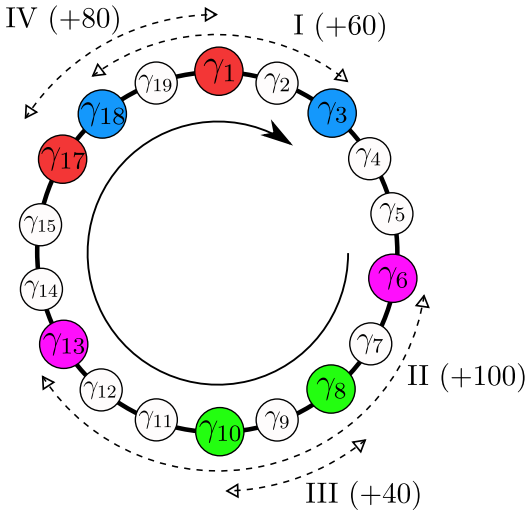

Figure 11 – Independent improving moves in the *best-independent-moves 2-opt* operator

Algorithm 4. Pseudo code of *best-independent-moves 2-opt* swap local search operator

Input: G // Weighted graph $G = (V, E)$
Input: D // Distance matrix
Input: Γ // A given solution
Input: $f(\Gamma)$ // Objective function value of solution Γ
Output: Γ // A local optimum

```
Function 2opt-swap-BIM(Γ, D):
    while ! terminationCriterion() do
        I = {}
        for i = 1 : |V| − 2 do
            for j = i + 2 : |V| do
                δ := d_{γi,γj} + d_{γi+1,γj+1} − d_{γi,γi+1} − d_{γj,γj+1}
                if δ < 0 then                                  // Archive improving moves
                    I[(γi, γj)] = δ
                end
            end
        end
        I := sort(I)                                  // Sort (γi, γj) pairs based on decreasing δ
        J := {}
        for (γi, γj) ∈ I do
            if (γi, γj) ∉ J then                              // Do moves without shared edges
                Γ' := swap(γi, γj, Γ)   // Swap the permutation between cities γi and
                                           γj
                f(Γ) := f(Γ) + δ
                J := J ∪ (γi, γj)
            end
        end
    end
    return Γ
```

- ***4-move insertion***: The basic insertion operator removes a random city from the permutation of cities and reinserts it into another position [JM97]. we propose a new insertion operator considering four types of moves: *forward-left* (FL), *forward-right* (FR), *backward-left* (BL), and *backward-right* (BR). Let's consider two sampled segments $\gamma_{i-1} \rightarrow \gamma_i \rightarrow \gamma_{i+1}$ and $\gamma_{j-1} \rightarrow \gamma_j \rightarrow \gamma_{j+1}$ of the tour, where the first segment is visited before the second segment. In addition, consider

that two cities γ_i and γ_j undergo the 4-move insertion operator. The four above-mentioned insertion moves produce four new segments of $\gamma_{j-1} \rightarrow \gamma_i \rightarrow \gamma_j \rightarrow \gamma_{j+1}$, $\gamma_{j-1} \rightarrow \gamma_j \rightarrow \gamma_i \rightarrow \gamma_{j+1}$, $\gamma_{i-1} \rightarrow \gamma_j \rightarrow \gamma_i \rightarrow \gamma_{i+1}$, and $\gamma_{i-1} \rightarrow \gamma_i \rightarrow \gamma_j \rightarrow \gamma_{i+1}$, respectively. Finally, the insertion move with the maximum improvement (i.e., the shortest travel distance) is applied to the solution. The pseudo code of the *4-move insertion* local search operator is provided as Algorithm 5.

Algorithm 5. Pseudo code of *4-move insertion* local search operator

Input: G // Weighted graph $G = (V, E)$
Input: D // Distance matrix
Input: Γ // A given solution
Input: $f(\Gamma)$ // Objective function value of solution Γ
Output: Γ // A local optimum
Function `4-move-insertion`(Γ, D)**:**
  while ! `terminationCriterion()` **do**
    for $i = 1 : |V| - 2$ **do**
      for $j = i + 2 : |V|$ **do**
        $\delta_{BL} := d_{\gamma_{i-1},\gamma_j} + d_{\gamma_i,\gamma_j} + d_{\gamma_{j-1},\gamma_{j+1}} - d_{\gamma_{i-1},\gamma_i} - d_{\gamma_{j-1},\gamma_j} - d_{j,\gamma_{j+1}}$
        $\delta_{BR} := d_{\gamma_i,\gamma_j} + d_{\gamma_{i+1},\gamma_j} + d_{\gamma_{j-1},\gamma_{j+1}} - d_{\gamma_i,\gamma_{i+1}} - d_{\gamma_{j-1},\gamma_j} - d_{j,\gamma_{j+1}}$
        $\delta_{FL} := d_{\gamma_{i-1},\gamma_{i+1}} + d_{\gamma_i,\gamma_{j-1}} + d_{\gamma_i,\gamma_j} - d_{\gamma_{i-1},\gamma_i} - d_{\gamma_i,\gamma_{i+1}} - d_{\gamma_{j-1},\gamma_j}$
        $\delta_{FR} := d_{\gamma_{i-1},\gamma_j} + d_{\gamma_i,\gamma_j} + d_{\gamma_i,\gamma_{j+1}} - d_{\gamma_{i-1},\gamma_i} - d_{\gamma_i,\gamma_{i+1}} - d_{j,\gamma_{j+1}}$
        $m^* := \arg\min_m \{\delta_m\}$ // Find best insertion move $m \in$ {BL,BR,FL,FR}
        if $\delta_{m^*} < 0$ **then** // Update solution during the local search
          $\Gamma' :=$ `insert`(i, j, m^*, Γ) // Apply insertion move m^* over cities γ_i and γ_j
          $f(\Gamma) := f(\Gamma) + \delta_{m^*}$
          break
        end
      end
    end
  end
  return Γ

As perturbation operators, we employ a set of three different types of operators including *Double-bridge*, *Shuffle-sequence*, and *Reversion-sequence* operators. The mechanism of these operators are provided as follows:

- ***Double-bridge***: This operator has been motivated to escape from local optima found by local search operators which perform only sequential moves such as the 2-opt, 3-opt, and Lin-Kernighan [MOF92]. The original Lin-Kernighan algorithm is based on doing sequential moves [Hel09]. For cities i and j to be exchanged, two cities should be adjacent. By other words, if two edges $(i, i+1)$ and $(j, j+1)$ are supposed to be removed, there should be the edge (i, j) in the route (i.e., i and j are adjacent). Such an exchange is called sequential. Double-bridge is the simplest non-sequential 4-opt move which cannot be undone by 2-opt, 3-opt and Lin-Kernighan, and thus, it avoids the search process being trapped in the local optimum generated by those local search operators. Using this operator, four edges are removed from the route of the cities and sub-routes are reconnected in order to obtain a non-sequential move so that Lin-Kernighan cannot undo. [MOF92]. For instance, consider two sampled non-overlapping segments $i-1 \rightarrow i \rightarrow i+1 \rightarrow i+2$ and $j-1 \rightarrow j \rightarrow j+1 \rightarrow j+2$. Applying the double-bridge operator over edges (bridges) $i \rightarrow i+1$ and $j \rightarrow j+1$ results two new segments as $i-1 \rightarrow j \rightarrow j+1 \rightarrow i+2$ and $j-1 \rightarrow i \rightarrow i+1 \rightarrow j+2$. The pseudo code of the *double-bridge*

perturbation operator is provided as Algorithm 6.

Algorithm 6. Pseudo code of *double-bridge* perturbation operator

```
Input: Γ                                                    // A given solution
Output: Γ                                                   // A perturbed solution
Function double-bridge(Γ):
    γ_i, γ_j := select(Γ)   // Select two cities γ_i and γ_j where the edges linked to
     them do not overlap
    Γ := exchange((γ_i, γ_{i+1}), (γ_j, γ_{j+1}), Γ)   // Exchange edges (γ_i, γ_{i+1}) and (γ_j, γ_{j+1})
    return Γ
```

- ***Shuffle-sequence***: The shuffle-sequence operator randomly selects a sequence of cities and shuffles (re-order) the selected sequence at random with the goal to perturb the solution [CWL19]. The pseudo code of the *shuffle-sequence* perturbation operator is provided as Algorithm 7.

Algorithm 7. Pseudo code of *shuffle-sequence* perturbation operator

```
Input: Γ                                                    // A given solution
Output: Γ                                                   // A perturbed optimum
Function shuffle-sequence(Γ):
    γ_i, γ_j := select(Γ)                  // Select two non-adjacent cities γ_i and γ_j
    Γ := shuffle(γ_i, γ_j, Γ)          // Shuffle the sequence between cities γ_i and γ_j
    return Γ
```

- ***Reversion-sequence***: The reversion-sequence operator randomly selects a sequence of cities and reverses the selected sequence [CWL19]. Noteworthy, a reversion-sequence move is equal to a single 2-opt move, no matter what gain (positive or negative) the move yields. The pseudo code of the *reversion-sequence* perturbation operator is provided as Algorithm 8.

Algorithm 8. Pseudo code of *reversion-sequence* perturbation operator

```
Input: Γ                                                    // A given solution
Output: Γ                                                   // A perturbed optimum
Function reversion-sequence(Γ):
    i, j := select(Γ)                        // Select two non-adjacent cities i and j
    Γ := reverse(i, j, Γ)                  // Reverse the sequence between cities i and j
    return Γ
```

It is worth mentioning that since the goal of perturbation operators is to perturb the solution to escape from a local optimum without losing many of the good properties of the current solution, in the QILS framework, the perturbation operators are performed only once.

4.2.2 QILS framework to select local search operators

As explained in the beginning of Chapter 4, the definition of states and actions depends on the goal of using Q-learning. In this section, the goal of Q-learning is to select the most appropriate local search operator at each step of the search process. Accordingly, we define the set of actions, states, and the reward function as follows:

Set of actions – The set of actions are defined as different local search operators that can be selected and applied at each step of the search process. The actions are going

to be learned using the Q-learning algorithm. Accordingly, in the QILS framework, the set of actions A consists of the *basic 2-opt*, *best-independent-moves 2-opt*, and *four-move insertion* operators which results in a set of actions of size three.

Set of states – In Q-learning, the application of any action is linked to the status of the environment, which is represented by a set of states. In this section, we aim to learn which operator is most appropriate to apply after applying the current operator. In other words, we aim to learn the sequence of local search operators. Accordingly, we define the set of states as the sequence of last k selected local search operators. For the case where k is equal to 1, the set of states becomes the same as the set of actions, including the local search operators.

Reward function – At the end of each episode, the performance of the employed local search operator (i.e., action) is evaluated based on a reward function. There are different mechanisms to calculate the reward. The most basic reward mechanism in the literature is to represent the reward as 0/1 values, depending on whether the operators have led to an improvement (i.e., reward equal to 1) or not (i.e., reward equal to 0) regardless of the amount of improvement [Nar03]. Alternatively, the reward value can be proportional to the amount of improvement that an operator obtains. In this section, we use the second mechanism, and the reward is computed by taking into account two improvements across the episode: the improvement in the current local optimum (i.e., local improvement) calculated as Equation (4.1), as well as the improvement of the best solution found (i.e., global improvement) calculated via Equation (4.2). These two improvements are then integrated into a single value as Equation (4.3), where η (1-η) represents the weights (importance) of a local (global) improvement. The higher the value of η, the higher the importance of local improvement. On the other hand, if the value of η is low, more importance is given to global improvement in comparison to local improvement, and a premature learning may occur when the best found solution is an isolated solution. In this way, no (or a very small) reward is assigned to the operator, even though good local improvements are achieved. Therefore, the value of η directly affects the learning procedure, and it should be carefully determined.

$$r_{local} = \frac{\max(F_b - F, 0)}{F_b} \tag{4.1}$$

$$r_{global} = \frac{\max(F_b^* - F^*, 0)}{F_b^*} \tag{4.2}$$

$$r = \eta \cdot r_{local} + (1 - \eta) \cdot r_{global} \tag{4.3}$$

If the employed operator is able to improve the current solution or the best solution found, it receives a positive reinforcement. Otherwise, the reward would be zero.

Other parameters – In this section, to generate the initial solution, we use the nearest-neighbor strategy, which is a greedy approach for initialization. we use double-bridge as a single perturbation operator. The `Acceptance(.)` function applies a *Metropolis acceptance* strategy [Met+53] that accepts all improved solutions and non-improved solutions with a probability of $\exp \frac{\Delta f}{T}$, where Δf is the difference between the objective function before and after applying the local search operator, and parameter T denotes

the cooling temperature which depends on the size of the instance and is computed as Equation (4.4), wherein τ is the temperature scale. The higher the value of T, the higher the chance to accept worse moves and vice versa.

$$T = \tau \cdot \sqrt{n} \tag{4.4}$$

4.2.3 QILS framework to select perturbation operators

In this section, the goal of Q-learning is to select the most appropriate perturbation operator at each step of the search process. Accordingly, we define the set of states, actions, and the reward function as follows:

Set of actions – In this algorithm, our goal is to select the best perturbation operator among different types of operator with the hope to enhance the exploration ability of the algorithm. Besides the type of the perturbation operators that has an impact on the exploration ability, another important feature is the intensity of perturbation, i.e., the number of replications of the perturbation operators. We incorporate both the operators' type and intensity into our action set to extend the action space to a more diverse set. Accordingly, we define an action as a tuple (P, R), where P is the type of the perturbation operator and R is the repetition number of the perturbation operator P. We consider the maximum number of repetitions R of double-bride, shuffle-sequence and reversion-sequence are considered equal to 3, 1, and 1, respectively.

Set of States – The main goal of applying perturbation operators in a MH is to help the search to escape from local optima and to converge faster toward (near-) optimal solution. Therefore, we relate the selection of perturbation operators to the status of the search, whether being trapped in local optima or not. Intuitively, when a MH gets trapped in a local optimum a more explorative perturbation operator is needed and in case of continuous improvement without a stuck in local optima a less perturbative operator may be more appropriate. Accordingly, we define the set of states as a binary set $S = \{0, 1\}$. State $s = 0$ represents a situation when the algorithm has been trapped in a local optimum and has failed to improve the current solution, and $s = 1$ represents a situation that the algorithm has been able to improve the current solution. The transition between states $s = 0$ and $s = 1$ happens as follows: I) if algorithm is in state $s = 0$ and no improvement happens, then $s' = 0$, II) if algorithm is in state $s = 0$ and an improvement happens, then $s' = 1$, III) if algorithm is in state $s = 1$ and an improvement happens, then $s' = 1$, and finally VI) if algorithm is in state $s = 1$ and no improvement happens, then $s' = 0$.

Reward function – The reward function is calculated in the same way using Equations (4.1) to (4.3).

Other parameters – In this section, we use the best-independent-moves 2-opt as a single local search operator.

4.3 Application to PFSP

In this section, we apply the QILS framework to solve the permutation flowshop scheduling problem. The framework is applied to select the perturbation operators of ILS with the hope to improve the exploration ability. Considering that the *insertion* operator (which we explain in the following) is the most effective local search operator for PFSP [FVRF17], we only include the insertion operator in the local search step and therefore,

there would be no operator selection in the local search step. However, we use a set of multiple perturbation operators and use the QILS framework to select among them. The set of problem-specific properties of the QILS framework including the set of search operators (local search and perturbation) for PFSP, and the properties of Q-learning such are defined below.

4.3.1 Set of search operators

Different operators exist for PFSP in the literature. The *insertion* operator is one of the most effective local search operators for PFSP, and it is the base of most developed operators for PFSP. As the perturbation operator, destruction-construction operator, whose base is the insertion operator, is one of the efficient operators.

Considerig that a solution to the PFSP is a permutation $\Pi := (\pi_1, \ldots, \pi_n) \in \mathfrak{S}(N)$ of the jobs, where $\pi_j \in N$ denotes the index of the job appearing at position j in the sequence Π, the mechanism of these operators have been explained in the following:

- ***Insertion***: The insertion operator removes a random element from the permutation of jobs and reinserts it into the best possible position which results in the best value of objective function. Most operators of PFSP rely on a speed-up technique called Taillard acceleration, introduced by Taillard in 1990 [Tai90]. This technique allows to reduce the computational complexity of evaluating the best possible position where a removed job can be reinserting and consequently speed up the search process. Whenever a job is being inserted in its best position, usually a considerable number of ties – different positions that provide the same minimum value of C_{max} – occurs. The mechanism employed to deal with (break) these ties has a large effect in the quality of the ïnĄnal solution obtained [FVF19]. Accordingly, different mechanisms have been introduced in the literature that prioritizes jobs based on some criteria [FVRF17]. The pseudo code of the *Insertion* operator is provided in Algorithm 9.

Algorithm 9. Pseudo code of *insertion* local search operator

```
   Input: Π                                                    // A given solution
   Output: Π                                                   // A local optimum
1  Function insertion(Π):
2      f* := f(Π)        // Remember C_max of the best solution found during the local
         search
3      Π* := Π           // Remember the best solution found during the local search
4      while ! terminationCriterion() do
5          Π^S := shuffle(Π)                                   // Shuffle sequence Π
6          for π ∈ Π^S do
7              k* := arg min_k {f(insert(Π, π, k))}
8              Π := insert(Π, π, k*)
9              if better(Π, Π*) then        // Update best solution found during the
                 local search
10                 f* := f(Π)
11                 Π* := Π
12             end
13         end
14     end
15     return Π*
```

- ***Destruction-construction***: This operator is a ruin-recreate operator where first,

a solution is partially ruined and then is rebuilt. Accordingly, this operator involves a constructive heuristic, wherein first, d jobs are randomly removed from the sequence of the current solution (*destruction phase*). Then, the extracted jobs are re-inserted one by one at their best position in the remaining partial sequence (*construction phase*). It is worth mentioning that d is a input parameter that needs to be determined a priori. The pseudo code of the *destruction-construction* operator is provided in Algorithm 10.

Algorithm 10. Pseudo code of *destruction-construction* perturbation operator

Input: Π // A given solution
Output: Π // A perturbed solution
Function destruction-construction(Π):
  // a. *Destruction phase*
  $D :=$ destruction(Π, d) // Remove d jobs from Π to form partial sequence $\Pi_{\setminus D}$
  $\Pi_{\setminus D} := \Pi \setminus D$
  // b. *Optional*: Local search (Algorithm 9) on partial sequence
  $\Pi_{\setminus D} :=$ insertion$(\Pi_{\setminus D})$
  // c. *Construction phase*
  for $\pi \in \Pi_{\setminus D}$ **do**
    $k^* := \arg\min_k \{f(\text{insert}(\Pi, \pi, k))\}$
    $\Pi :=$ insert(Π, π, k^*)
  end
  return Π

It is worth mentioning that in the QILS framework, the local search operators perform a descent-based search and continue until no more improvements are found.

4.3.2 QILS framework to select perturbation operators

In this section, a Q-learning algorithm is integrated into the perturbation mechanism to adaptively select perturbation operators during the search process. The motivation behind using Q-learning in the perturbation step lies on the fact that using the destruction-construction operator, the user needs to tune the value of d which controls the strength and intensity of perturbation a priori. As a result, the solution is perturbed similarly at each step of the search process. This mechanism considers a constant degree of exploration regardless of the search state, which limits the ability of the algorithm to escape from the local optima. To cope with this issue, an appropriate degree of exploration needs to be adjusted at different steps of the search process based on the status of the search.

Accordingly, we define the set of states, actions, and the reward function as follows:

Set of actions – The set of actions is defined as different perturbation operators with different strengths that can be applied at each step of the search process. More precisely, a set of different values of d in the *destruction-construction* operator is considered as the set of actions A represented as:

$$A = \{1, 2, \ldots, d_{max}\}, \tag{4.5}$$

where d_{max} is the maximum number of jobs to be removed in the *destruction* phase. The goal of considering different values of d is to provide a set of perturbation operators with different exploration strengths. Indeed, as the value of d increases, a higher number of jobs are removed from and re-inserted into the sequence of jobs, which leads to a higher degree of perturbation in the solution.

Set of States – We recall that in this work, the main goal of integrating Q-learning into the Iterated Greedy algorithm is to select the most appropriate perturbation operators at each step of the search process, with the hope to improve the exploration ability of the algorithm. Therefore, similar to TSP, we relate the selection of perturbation operators to the status of the search whether being trapped in local optima or not. In this regard, we define the set of states as a binary set $S = \{0, 1\}$, where $s, s' \in S$. State $s = 1$ is attained when the current perturbation operator has been successful in the recent episode to bring out the solution from the local optimum. Otherwise, when state $s = 0$, the operator has failed and a perturbation operator with different strength is required.

Reward function – The reward function is calculated in the same way using Equations (4.1) to (4.3).

Other parameters – To generate initial solutions for PFSP, we use a heuristic called Nawaz, Enscore and Ham (NEH) [NEJH83]. This heuristic is one of the best methods of initial solution generation for solving PFSP by meta-heuristic algorithms [FVF19; FVMPF20]. The pseudo code of NEH heuritic is given in Algorithm 11.

Algorithm 11. Pseudo code of the NEH heuristic

Output: Π // A complete sequence of jobs
Function `NEH()`:
  $\Pi := \emptyset$ // Start by an empty sequence Π
  $\Pi^S :=$ `sort()` // Sort jobs $j \in N$ in descending order based on $\sum_{i \in M} p_{ij}$
  $\Pi := \Pi \cup \pi_1$ // π_1: the job in the first position of Π^S
  $\Pi^S := \Pi^S \setminus \{\pi_1\}$ // remove π_1 from Π^S
  while $\Pi^S \neq \emptyset$ **do**
    $k^* := \arg\min_k \{f(\texttt{insert}(\Pi, \pi_1, k))\}$ // Find the best position k^* in Π to insert π_1
      // insert(Π, π_j, k): inserts π_j in kth position of sequence Π ($k \in K, |K| = |\Pi| + 1$)
    $\Pi :=$ `insert`(Π, π_1, k^*)
    $\Pi :=$ `localSearch`(Π) // Optional: apply local search on partial sequence Π
    $\Pi^S := \Pi^S \setminus \{\pi_1\}$
  end
  return Π

To deal with the ties that occurs during the search, we employ the tie-breaking mechanism proposed by [FVF14] (i.e., during initial solution generation, construction phase, and local search). This tie-breaking mechanism prioritizes jobs with the lowest sum of the idle times in the sequence.

To accept new local optima, this work uses the Metropolis acceptance strategy [Met+53], which allows non-improving solutions to be accepted with a probability of $\texttt{exp}(\frac{C_{max}(\Pi) - C_{max}(\Pi')}{T})$, where T is a cooling temperature parameter which depends on the size of the instance and is computed as Equation (4.6), wherein τ is the temperature scale [RS07].

$$T = \tau \cdot \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}}{n \cdot m \cdot 10} \tag{4.6}$$

4.4 Experimental design

This section designs comprehensive experiments to investigate the performance of the QILS framework. The experiments are conducted on a wide variety of instances to answer four research questions:

- Does incorporating multiple local search (perturbation) operators enhance the exploitation (exploration) ability and the performance of ILS and how much?
- Does employing Q-learning to select appropriate operators improve the performance of ILS and how much?
- How competitive is the proposed QILS framework in comparison to the state-of-the-art algorithms from the literature?
- How does the proposed QILS framework automatically adapt the operators to the problem instance at hand?

4.4.1 Dataset

The computational experiments are conducted on several well-known datasets of TSP and PFSP instances with different sizes.

TSP – For TSP, we have used a set of randomly selected symmetric TSP instances from the TSPLIB library [Tsp] with different number of cities ranging from 50 to 2150 cities.

PFSP – For PFSP, we have used three well-known datasets, namely *Taillard*, *VRF-hard-small*, and *VRF-hard-large* datasets. The characteristics of each dataset are as follows:

- *Taillard* dataset [Tai93] is a well-known and common dataset used by almost all studies to evaluate the performance of their algorithms in solving PFSP. This dataset contains 120 instances categorized into 12 sets of instances ranging from 20 jobs with 5 machines to 500 jobs with 20 machines.
- *VRF-hard-small* dataset [VRF15] is a well-known dataset developed in 2015, with the aim to propose a benchmark dataset containing hard instances with more discriminant power than the most common benchmark from the literature. In order to generate hard instances, they have done exhaustive experimental procedure, generating thousands of instances and selecting the hardest ones based on the gap computed by different algorithms. This dataset includes 240 instances categorized into 24 sets of instances ranging from 10 jobs and 5 machines to 60 jobs and 20 machines.
- *VRF-hard-large* dataset [VRF15] includes 240 hard large instances categorized into 24 sets of instances ranging from 100 jobs and 20 machines to 800 jobs and 60 machines.

4.4.2 Benchmark comparison

In order to answer the four above-mentioned questions, two major experimental phases are designed. The first phase aims at answering the first two questions, by comparing the performance of the proposed QILS framework with two algorithms: ILS algorithms

with an individual local search (perturbation) operator denoted as IILS and an ILS algorithm with the same set of local search (perturbation) operators as QILS selected uniformly and randomly denoted as RILS. The stopping criterion of all algorithms is fixed to a predetermined number of iterations. Let's consider the maximum number of iterations for the `while` loop of QILS, RILS, and IILS as I_{QILS}, I_{RILS}, and I_{IILS}, respectively. To determine a suitable number of iterations (i.e., not too small to lose further improvements and not too large to avoid unnecessary computational efforts) for each instance set, in a trial and error way, all algorithms were executed and once no significant improvement is observed (i.e., the improvement over the last 10% of the search process divided by the total improvement so far is less than 0.01), the algorithms were stopped. Next, the maximum number of total iterations among all algorithms (i.e., $\max\{I_{QILS} \times E, I_{RILS}, I_{IILS}\}$) was considered as the stopping criterion for all algorithms.

The second phase aims at answering the third question. It is addressed by comparing QILS to the state-of-the-art algorithms from the literature, which are the most recent and relevant algorithms to our work. Since in this phase the aim is to evaluate the competitiveness of the proposed QILS framework, the stopping criterion for all algorithms is fixed as a limited CPU time of $T = \frac{nm}{2}t$ milliseconds (ms) where t is a scale ($t \in \{60, 90, 120\}$) [RS07].

Finally, the results of QILS are used to answer the fourth research question. In this regard, we extract information on the contribution of each operator to the overall improvement from the initial solution to the final best found solution. In addition, it is shown how different operators have been applied at each step of the search process.

For TSP, the experiments have been limited to phase 1, comparison with the non-learning versions of QILS, since the primary goal of this application was to investigate the effectiveness of incorporating Q-learning into ILS for operator selection. Based on the primary results and considering the availability of many efficient algorithms in the literature for solving TSP (i.e., exact and approximate algorithms), comparison with these state-of-the-art algorithms has not been conducted. Actually, the application of the proposed QILS framework on TSP is just to validate the interest of integrating the Q-learning in the ILS algorithm, and not indeed obtain the state-of-the-art behavior. Furthermore, the ILS algorithm is not itself the state-of-the-art algorithm for solving TSP instances; hence, any improvement in ILS using Q-learning does not necessarily yield state-of-the-art behavior.

To summarize, QILS is compared to a set of benchmark algorithms for TSP and PFSP, which are described as follows:

TSP – For TSP, we compare QILS with non-learning versions of QILS as follows:

- ILS algorithms with a single local search (perturbation) operator to compare with the QILS of Section 4.2.2 (4.2.3)
- ILS algorithm with the same set of local search (perturbation) operators as QILS selected uniformly and randomly to compare with the QILS of Section 4.2.2 (4.2.3)

PFSP – For PFSP, we compare QILS with non-learning versions of QILS as well as a set of seven state-of-the-art algorithms from the literature:

- ILS algorithms with a single perturbation operator (i.e., with fixed values of d
- ILS algorithm with the same set of perturbation operators (i.e., with different values of d) as QILS selected uniformly and randomly

- Seven state-of-the-art algorithms from the literature proposed by [RS07], [PTL08], [FVF14], [DLPS17], [Kiz+19], [PS19], and [FVF19] which are the most recent and relevant algorithms to our work.

It is worth mentioning that, we implemented all the benchmark algorithms including the state-of-the-art algorithms in the literature under the same conditions; they are coded in the same programming language sharing the same libraries, functions, and data structures and executed on the same computer environment with the same configurations. All algorithms are coded in Python 3.7 and all experiments are carried out on four servers each containing four Intel XEON processors with 5GB of RAM memory running at 2.3 GHz. It should be noted that only one processor was used to carry out the experiments, and no parallel programming technique is employed. To ensure the reproducibility of the results, the source codes used to perform the subsequent experiments have been put online [1].

4.4.3 Performance comparison metrics

Considering the stochastic nature of the algorithms and to investigate the robustness of the QILS framework for finding the (near-) optimal solutions, each algorithm has been run 30 independent times for each instance and the results are averaged. Three key performance comparison metrics are used to compare the performance of the algorithms:

- Average Relative Percentage Deviation (ARPD): To evaluate the quality of the solutions, we use ARPD from the proven optimum (if available) or from the best-known solution in the literature. The Relative Percentage Deviation (RPD) of algorithm j for instance i, $RPD_{i,j}$, is calculated as Equation 4.7, where $f_{i,j}(S)$ is the objective function value of the solution obtained by algorithm j for instance i, and $f_i^*(S)$ is the objective function value of the optimal solution (or best-known solution) for that instance. The best-known solutions of each instance of TSP and PFSP as well as the best found solutions of the QILS framework are reported in Tables A.2 to A.4 in Appendix A.2.

$$RPD_{i,j} = \frac{f_{i,j}(S) - f_i^*(S)}{f_i^*(S)} \times 100 \tag{4.7}$$

- Average computational time: To evaluate the competitiveness and complexity of the proposed algorithm with respect to the other algorithms, we use the average time to achieve the final solution.
- Standard deviation: To evaluate the robustness of the solutions, we use the standard deviation that measures how disperse are the obtained solution for different executions of the same setting with respect to the mean value.
- Convergence behavior: The convergence behavior of an algorithm measures how fast (i.e., when/ at which iteration) the algorithm converges to the best found (optimal) solution.

4.4.4 Statistical test

Statistical tests are widely used to investigate whether there is a statistically significant difference between a new proposed method over the existing methods [Der+11]. Statistical test are categorized into parametric and non-parametric methods, depending on the type of data employed. Parametric methods are based on a set of assumptions with

1. Source codes: `https://osf.io/x7y8d/?view_only=a2a33ed1a10441e2abf102beb14bbb00`

respect to the employed data such as the independence, normality, and homoscedasticity. When the previous assumptions cannot be satisfied, a non-parametric test can be used.

In this thesis, since the difference between two algorithms' means cannot be assumed to be normally distributed, we employ a non-parametric test, the Wilcoxon signed rank test [Wil92] with 95% confidence level for statistical comparisons. This test investigates the difference between paired scores obtained by two compared algorithms, where the null hypothesis indicates no significant difference between the median of two algorithms. If the p-value is less than 0.05, the null hypothesis is rejected in favor of the alternative hypothesis that explains a significant difference between the median of the two algorithms. We perform the Wilcoxon signed rank test for each pairwise comparison between the QILS framework and another algorithm (IILS, RILS, etc.) to see whether the QILS framework offers a significant improvement over the existing algorithms or not. Throughout the experiments of Section 4.5, we provide the minimum, average, and maximum of statistically significant p-values.

4.4.5 Parameters tuning

Since the performance of meta-heuristics are widely affected by their parameter values, they need to be tuned carefully. Indeed, the values of parameters directly affect the exploration/exploitation abilities of the algorithm to find (near-) optimal solutions. In this paper, the values of parameters are tuned by the Response Surface Methodology (RSM) [BW51]. RSM is a regression-based optimization method that identifies a dependent variable (i.e., response value) as a function of independent variables through a set of experiments. Among various design of experiment methods, we have used the $3k$ factorial Box-Behnken design due to its advantages in requiring fewer experiments and having high efficiency [BB60]. To use the RSM, first, the significant parameters are identified. Accordingly, we identify seven effective parameters for the proposed QILS framework including τ (temperature scale for the Metropolis acceptance strategy), ϵ (probability of exploring new actions), β (ϵ-decay rate), α (learning rate), γ (discount factor), E (episode size), and η (local/global improvement weight). In a $3k$ factorial Box-Behnken design, three levels are considered for each variable, coded by -1, 0, and $+1$, which represent the min, mean, and max levels of the variables, respectively. To transform the uncoded values of variables into these standardized levels, we use Equation 4.8, where X_i is the coded value of the i^{th} independent variable, x_i is its uncoded value, and x_{max} and x_{min} are the maximum and minimum levels of the i^{th} uncoded variable.

$$X_i = \frac{x_i - \frac{x_{max}+x_{min}}{2}}{\frac{x_{max}-x_{min}}{2}} \tag{4.8}$$

A seven-variable Box-Behnken design is performed, which results in 62 experiments [BB60]. To perform the experiments, for TSP, a set of instances with different sizes are selected randomly from the TSPLIB library. For PFSP, a set of 60 instances are selected randomly from each set of instances in the benchmarks (i.e., 12 instances from *Taillard*, 24 instances from *VRF-hard-small* and 24 instances from *VRF-hard-large*) and the stopping criterion is considered as $T = \frac{nm}{2}60$ ms. Using RSM and Box-Behnken design, the levels and the tuned values of the significant parameters of the proposed QILS framework for TSP and PFSP are shown in Table 13. In these tables, the levels "-1" and "$+1$ of the parameters have been identified either logically (e.g., the level $+1$ of parameters τ, ϵ, β, α, γ and the level "-1" of parameter E) or based on the literature. Furthermore, to provide a more trustworthy parameter tuning, the identified ranges of

the parameters from the literature are enlarged with a margin of 10%, if possible. The level "0" of the parameters then represents the middle of the identified ranges. Finally, the ranges of parameters η is fixed to the interval $[0.2, 0.8]$, to provide a balance between the inclusion of the local and global improvements in the calculated reward values.

Table 13 – Parameters of QILS, their corresponding levels and tuned values

Parameter	Notation	Levels			Tuned value	
		-1	0	+1	TSP	PFSP
Temperature scale	τ	0.5	0.75	1	1	0.7
Epsilon-greedy	ϵ	0.7	0.85	1	0.8	0.8
Epsilon-decay	β	0.990	0.995	1	0.999	0.996
Learning rate	α	0.5	0.75	1	0.6	0.6
Discount factor	γ	0.7	0.85	1	0.5	0.8
Size of episode	E	1	4	7	1	6
Local/global improvement weight	η	0.2	0.5	0.8	0	3

According to Section 4.3.2, for PFSP, we apply the QILS framework to select the best perturbation operator among a set of available operators. In this regard, destruction-construction operator with different values of d are considered as a set of available perturbation operators. In destruction-construction operator, d is a parameter related to the number of jobs to remove in destruction phase. In order to determine the set of perturbation operators, we need to determine the value of d. According to the literature, IILS_d with lower values of d provide better performance [RS07; DLPS17; FVF19]. However, we perform additional experiments based on different values of d (i.e., $d = 1, 2, ..., 10$) with a predetermined number of iterations (see Figure 15) to investigate this assertion. Figure 12 shows the average performance (i.e., Average Relative Percentage Deviation - ARPD (%)) of IILS_d for different values of d on *Taillard* dataset. As it can be seen, the best average performance belongs to IILS_2 and for greater values of d, as the value of d increases the average performance of IILS_d degrades. Additionally, we investigate whether the difference between IILS_d ($d = 1, 2, ..., 10$) is statistically significant. For this aim, we employ the Wilcoxon signed rank test [Wil92] with 95% confidence level (see Section 4.4.4 for more details).

We observe that $\text{IILS}_{d\geq 4}$ both on average and for each instance set is statistically dominated by at least one of other three IILS_1, IILS_2, and IILS_3. Furthermore, no dominance observed among IILS_1, IILS_2, and IILS_3 over different instance sets. Accordingly, we incorporate three perturbation operators with $d = 1$, $d = 2$, and $d = 3$ (i.e., $d_{max} = 3$) into QILS and RILS.

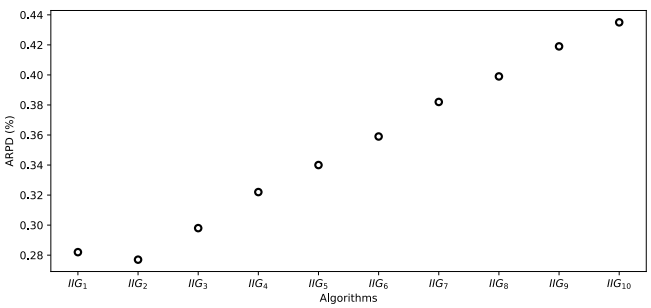

Figure 12 – Performance comparison of IILS_d based on ARPD (%) for *Taillard* dataset

4.5 Computational results

4.5.1 Application of QILS to TSP

In this section, first, we compare the performance of RILS over IILS to investigate how employing multiple perturbation operators affects the exploration ability of an ILS algorithm. Then, we compare the performance of QILS with RILS to demonstrate the effectiveness of Q-learning in operator selection over random selection. It is worth mentioning that the following results are only to show the interest of integrating Q-learning into ILS to solve the well-known TSP. Hence, these results are not competitive compared to the state-of-the-art.

Comparison based on optimality gap – Tables 14 and 15 show the performance of QILS for both local search and perturbation operator selection and their corresponding RILS and IILS algorithms for different instances of TSPLIB. In each of these tables, column "AvS" reports the ARPD (%) across 30 solutions (i.e., 30 independent runs for each instance) obtained by each algorithm for each instance. In addition, column "BS" shows the RPD (%) of the best solutions (i.e., the best solution among 30 runs) obtained by each algorithm for each instance. Finally, column "T (s)" indicates the average CPU time in seconds across all 30 runs of each instance.

Table 14 indicates that QILS for local search operator selection is able to find optimal solution in both small- and medium-sized instances, and it is able to find near-optimal solutions with an optimality gap of 3.83% for the largest instance with 2152 cities. By looking at the "AvS" and the "BS" results, it can be seen that QILS has produced small gaps over all 30 executions. In terms of the CPU time, the higher the size of the instance, the higher the CPU time of the algorithm. By looking at the column "T", it can be seen how expensive certain instances are in terms of CPU time. Some observations from Table 14 can also be generalized to the results of Table 15. Besides the zero optimality gap for small- and medium-sized instances, QILS for perturbation operator selection is even able to find optimal solution for some large-sized instances up to 300 cities. For larger instances, small gaps have been also reported with an optimality gap of 3.94% for the largest instance with 2152 cities.

By comparing RILS with IILS in both Tables 14 and 15, it can be concluded that RILS outperforms IILS in almost all instances. These results answer the first research question and imply that incorporating multiple perturbation operators into the ILS algorithm significantly improves the exploration ability of the ILS algorithm when solving TSP instances. We now analyze the benefits of using a Q-learning approach for operator selection over a random one, which provides the answer to the second research question. Tables 14 and 15 show that QILS can reach better solutions in terms of both AvS and BS in almost all instance sets comparing to RILS. This highlights the outperformance of the proposed QILS framework in terms of the optimality gap. This observation illustrates the efficiency of integrating the knowledge from the Q-learning algorithm into the operator selection mechanism of the ILS algorithm.

Table 14 – Performance comparison of QILS for local search operator selection with the corresponding RILS, and IILS in TSP application. The best values for each set of instances among different algorithms have been highlighted in gray. Furthermore, bold values indicate results that are not statistically distinguishable from results of the QILS algorithm. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.003, 0.072)$.

Instance	Algorithms														
	$IILS_1$			$IILS_2$			$IILS_3$			RILS			QILS		
	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
berlin52	0.204	0	0.610	0.353	0	0.900	0.752	0	0.600	0.164	0	1.100	0	0	0.663
st70	0.993	0	6.700	0.874	0	6.100	1.037	0	7.010	0.800	0	6.400	**0.037**	0	9.883
kroA100	0.684	0.108	8.200	0.432	0.301	11.500	2.180	0.601	10.400	0.349	0	11.600	**0.005**	0	10.200
rd100	1.707	0.695	16.200	1.808	0.506	14.600	3.932	3.173	18.900	1.145	0.430	16.800	**0.173**	0	25.933
lin105	0.869	0	13.300	0.846	0	15.100	2.288	1.015	16.500	0.689	0	13.600	**0**	0	15.247
pr124	0.491	0.097	24.500	0.281	0	31.800	1.882	0.935	34.500	0.385	0	30.300	**0.004**	0	34.116
ch130	2.399	1.408	58.300	1.039	0.131	60.500	3.666	3.159	50.700	1.468	0.229	65.600	**0.546**	0.262	73.380
ch150	1.656	0.490	43.500	1.765	1.103	47.500	4.519	3.676	43.400	1.419	0.322	46.200	**0.465**	0.077	50.796
u159	1.713	1.440	61.200	1.262	0	66.000	2.548	2.405	70.600	1.702	0	65.100	**0.492**	0	55.360
d198	0.650	0.234	170.300	0.772	0.190	188.200	2.260	1.141	201.400	0.699	0.057	185.100	**0.263**	0.165	220.863
kroA200	1.158	0.661	184.400	0.718	0.054	214.100	2.329	1.583	214.100	0.883	0.129	221.900	**0.352**	0.051	237.510
ts225	0.811	0.187	62.000	0.968	0.696	47.000	2.749	0.187	46.900	0.723	0	46.900	**0.013**	0	48.030
pr264	1.139	0.092	180.600	1.076	0.263	177.600	5.709	3.446	170.600	1.409	0.794	170.100	**0.402**	0	207.061
a280	2.443	0.814	314.500	2.869	1.978	347.200	6.126	4.847	333.200	2.559	1.590	362.200	**0.587**	0	396.827
pr299	2.491	1.836	605.300	2.075	1.828	646.700	5.493	3.779	644.900	2.654	2.139	618.800	**0.805**	0.151	704.277
lin318	2.049	1.278	591.300	2.153	1.604	605.600	3.048	2.563	632.500	2.144	1.549	604.900	**1.559**	0.895	688.655
fl417	2.712	1.113	683.700	2.900	1.846	701.900	7.984	5.843	706.300	2.378	1.374	738.900	**0.966**	0.430	796.136
pr439	4.406	3.553	709.300	4.029	3.622	690.700	6.281	5.441	615.500	3.848	2.174	722.100	**2.308**	0.755	778.177
pcb442	2.932	2.030	681.600	2.992	1.798	661.300	5.577	4.984	687.500	2.457	1.587	698.000	**1.568**	1.061	758.390
d493	3.020	1.617	594.900	3.657	2.603	648.400	7.051	5.483	604.200	3.007	2.311	659.500	**2.135**	1.451	643.362
vm1084	5.266	3.703	2186.200	5.398	4.217	1953.125	12.399	12.112	2006.400	5.082	3.610	2082.000	**4.217**	3.025	2012.244
d1291	5.177	3.335	1946.600	5.658	3.903	1817.900	10.422	10.090	1911.900	5.498	2.248	1808.100	**4.295**	3.173	1835.273
u1817	6.278	5.409	596.500	7.812	6.250	667.900	10.801	10.281	771.841	7.153	4.472	765.800	**4.858**	4.142	664.385
u2152	6.162	4.825	778.300	7.416	6.001	880.000	11.447	10.701	956.400	7.412	4.493	850.500	**4.841**	3.836	919.772

Table 15 – Performance comparison of QILS for perturbation operator selection with the corresponding RILS, and IILS for TSP application. The best values for each set of instances among different algorithms have been highlighted in gray. Furthermore, bold values indicate results that are not statistically distinguishable from results of the QILS algorithm. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.001, 0.048)$.

Instance	Algorithms														
	$IILS_1$			$IILS_2$			$IILS_3$			RILS			QILS		
	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
berlin52	0.167	0	0.7	0.239	0	0.6	0.219	0	0.7	0.003	0.000	0.2	0	0	0.2
st70	0.770	0.148	4.2	0.415	0	4.1	0.815	0.148	3.1	0.681	0.148	3.2	**0.109**	0	4.5
kroA100	0.583	0.000	7.7	0.328	0	8.1	0.438	0	8.7	0.227	0	8.1	**0**	0	7.3
rd100	0.885	0.695	11.1	0.927	0	9.8	1.088	0	10.2	0.714	0	10.6	**0.201**	0	14.2
lin105	0.977	0	7.8	0.759	0	9.3	0.462	0.000	8.7	0.174	0	5.6	**0**	0	4.5
pr124	0.571	0	11.9	0.377	0	10.7	0.469	0.078	12.3	0.221	0	8.9	**0**	0	8.7
ch130	1.434	0.802	27.4	1.126	0.245	24.4	1.198	0.769	26.6	1.129	0.720	25.2	**0.359**	0	40.4
ch150	2.106	1.486	23.8	1.385	0.475	24.6	1.544	0.659	27.0	0.928	0.414	19.6	**0.374**	0	33.2
u159	1.435	0.803	44.2	0.586	0.000	42.8	0.507	0.000	34.1	0.264	0.000	33.1	**0.112**	0	42.9
d198	1.363	0.976	122.8	0.887	0.253	121.3	0.749	0.418	106.5	0.368	0.228	97.4	**0.180**	0.057	147.5
kroA200	1.282	0.855	153.4	0.420	0.000	148.3	0.719	0.320	157.2	0.344	0.000	211.6	**0.150**	0.000	194.8
ts225	1.043	0.641	84.1	0.585	0.285	95.2	0.531	0.364	108.1	0.077	0.000	95.7	**0.002**	0	108.9
pr264	1.257	0.995	107.7	0.796	0.311	101.4	0.500	0.033	112.9	0.390	0.033	120.8	**0.172**	0	135.8
a280	4.044	3.024	216.3	2.497	0.310	214.4	2.389	0.698	221.2	2.212	0.388	266.2	**0.498**	0	276.5
pr299	3.741	2.706	403.1	2.336	0.689	400.7	2.557	1.384	380.8	2.495	1.600	451.2	**0.274**	0	440.7
lin318	3.755	2.927	407.3	2.626	1.394	425.1	2.527	1.554	403.0	2.225	1.651	492.1	**0.795**	0.302	479.6
rd400	4.651	3.802	475.7	3.432	2.375	486.0	3.491	1.983	475.0	2.927	0.962	541.5	**1.461**	0.825	519.4
fl417	4.235	1.425	457.9	2.535	1.239	467.1	2.332	0.995	422.8	1.514	0.590	448.4	**0.339**	0.211	553.0
pr439	3.157	1.791	602.8	3.215	0.889	501.2	3.287	1.034	499.5	3.057	1.959	640.2	**1.392**	0.438	528.2
pcb442	3.313	2.578	616.6	3.161	2.407	502.6	3.475	2.724	511.7	2.385	1.804	614.7	**1.294**	0.640	521.6
d493	4.511	3.437	527.4	3.709	2.437	511.8	3.588	2.774	537.3	2.776	1.703	604.6	**1.499**	0.923	587.8
vm1084	5.887	4.984	2073.4	5.172	4.011	2039.7	5.675	4.271	1845.6	4.911	3.458	1822.8	**3.717**	3.061	2608.3
d1291	6.555	4.092	2356.5	5.799	3.870	1605.3	6.129	4.335	1541.2	5.547	2.327	1580.7	**3.418**	2.281	2060.1
u1817	7.685	6.250	1232.7	7.220	6.087	1323.2	7.354	6.250	1488.8	6.003	3.844	942.1	**4.366**	3.449	1416.0
u2152	7.008	6.001	1169.3	7.421	6.001	1487.9	7.378	6.001	1306.0	6.357	4.123	1261.3	**4.531**	3.947	1022.3

4.5.2 Application of the QILS framework to PFSP

4.5.2.1 Phase 1: comparison between QILS, RILS, and IILS algorithms

In this section, first, we compare the performance of RILS over IILS to investigate how employing multiple perturbation operators affects the exploration ability of an ILS algorithm. Then, we compare the performance of the QILS framework with RILS to demonstrate the effectiveness of Q-learning in operator selection over random selection.

Comparison based on optimality gap – Tables 16, 17 18 show the performance of QILS, RILS, and four IILS$_d$ for different instance sets of *Taillard*, *VRF-hard-small*, and *VRF-hard-large* datasets, respectively. In each of these tables, column "AvS" reports the ARPD (%) across 300 solutions (i.e., 10 instances of each size and 30 independent runs for each instance) obtained by each algorithm for each instance set with a size of n jobs and m machines. In addition, column "BS" shows the ARPD (%) across 10 best solutions (i.e., the best solution among 30 runs) obtained by each algorithm for each instance set. Finally, column "T (s)" indicates the average CPU time in seconds across all 300 runs of each instance set. The negative values in columns "AvS" and "BS" indicate obtaining better solutions compared to the upper bound of the best-known solutions in the literature.

By comparing RILS with IILS$_d$ in all tables, it can be concluded that RILS outperforms IILS$_d$ in almost all sets of instances. The statistical analyses conducted using Wilcoxon signed rank test also shows a significant difference between RILS and each of the IILS$_d$ on average. These results answer the first research question and imply that incorporating multiple perturbation operators into the ILS algorithm significantly improves the exploration ability of the ILS algorithm when solving PFSP instances.

We now analyze the benefits of using a Q-learning approach for operator selection over a random one, which provides the answer to the second research question. Tables 16 to 18 show that QILS can reach better solutions in terms of both AvS and BS in almost all instance sets comparing to RILS. Focusing on *Taillard* dataset, as the size of the instance sets increases, the outperformance of QILS becomes clearer. Regarding the *VRF-hard-small* dataset, the difference between algorithms is not that significant, since all algorithms are able to find good solutions for such small instances. The results on the *VRF-hard-large* dataset clearly show that the proposed perturbation mechanism is a key component for the success of the ILS algorithm that considerably enhances the exploration ability of the algorithm. The observed behavior has also been statistically verified.

Let us now analyze the performance of QILS, RILS, and IILS$_d$ in more details. Due to space limitations, we only focus on the *Taillard* dataset, though the same overall conclusions have been obtained for the other two datasets. To illustrate the robustness of QILS, Figure 13 depicts the boxplots of RPD (%) of 30 independent runs of all algorithms for *Taillard* dataset. As it can be seen, QILS obtains better median and mean values, and it even shows more robust behavior (i.e., lower standard deviation) compared to RILS and IILS$_d$, particularly for large-sized instance sets. For the small-sized instance sets, QILS exhibits a highly robust performance as it is able to reach the optimal solution for "tai_20_5" and a very small RPD for "tai_50_5" and "tai_100_5".

Table 16 – Performance comparison of QILS, RILS and IILS$_d$ on *Taillard* dataset. The best values for each set of instances among different algorithms have been highlighted in gray. Furthermore, bold values indicate results that are not statistically distinguishable from results of the QILS algorithm. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.004, 0.041)$.

Instance		Algorithms																	
set		IILS$_1$			IILS$_2$			IILS$_3$			IILS$_4$			RILS			QILS		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
20	5	0.041	0.041	1.2	0.041	0.041	1.4	0.039	0	1.5	0.041	0.041	1.6	0.039	0	1.7	0.036	0	1.8
20	10	0.066	0	1.5	**0.027**	0	1.8	**0.034**	0	1.9	**0.034**	0	2.0	**0.029**	0	2.0	**0.027**	0	2.2
20	20	0.054	0	4.1	**0.011**	0	4.6	**0.009**	0	4.9	**0.012**	0	5.0	**0.010**	0	5.0	**0.007**	0	5.3
50	5	0.002	0	5.7	0.003	0	6.6	0.002	0	7.0	0.002	0	7.1	0.002	0	6.9	0.002	0	7.3
50	10	0.436	0.288	18.7	0.440	0.290	22.9	0.487	0.307	25.6	0.544	0.348	26.4	0.428	0.279	24.0	**0.401**	0.255	24.6
50	20	**0.546**	0.272	37.8	**0.555**	0.315	45.6	0.610	0.333	50.6	0.697	0.400	52.5	**0.556**	0.304	45.9	**0.538**	0.239	44.7
100	5	0.009	0.008	18.1	0.009	0.008	19.9	**0.008**	0.008	21.0	0.010	0.008	21.6	**0.008**	0.008	20.6	**0.008**	0.008	21.7
100	10	0.085	0.030	40.6	0.094	0.019	48.4	0.118	0.032	51.7	0.130	0.028	53.2	0.079	0.021	47.6	**0.052**	0.016	48.9
100	20	0.714	0.432	156.6	0.761	0.479	192.9	0.830	0.572	211.9	0.909	0.647	225.7	0.671	0.411	192.4	**0.608**	0.315	185.3
200	10	0.057	0.037	65.5	0.050	0.032	77.5	0.059	0.035	84.6	0.060	0.030	88.5	0.046	0.033	74.9	**0.043**	0.024	76.9
200	20	0.926	0.642	295.9	0.910	0.618	377.9	0.963	0.682	419.4	0.989	0.723	445.5	0.749	0.493	349.5	**0.662**	0.455	334.7
500	20	0.485	0.352	916.4	0.466	0.334	1102	0.459	0.333	1144	0.472	0.341	1159	0.338	0.218	913.0	**0.288**	0.202	812.9
Average		0.285	0.175	130.2	0.281	0.178	158.5	0.302	0.192	168.7	0.325	0.214	174.1	0.246	0.147	140.3	**0.223**	0.126	130.5

Table 17 – Performance comparison of QILS, RILS and IILS$_d$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.005, 0.040)$.

Instance		Algorithms																	
set		IILS$_1$			IILS$_2$			IILS$_3$			IILS$_4$			RILS			QILS		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
10	5	0	0	0.0	0	0	0.0	0	0	0.0	0	0	0.0	0	0	0.1	0	0	0.1
10	10	0.006	0	0.1	0	0	0.1	0	0	0.1	0	0	0.1	0	0	0.1	0	0	0.1
10	15	0.037	0	0.1	**0.001**	0	0.1	**0**	0	0.1	**0.001**	0	0.1	**0**	0	0.1	**0**	0	0.1
10	20	0.058	0	0.1	0.019	0	0.1	**0.001**	0	0.1	0.004	0	0.1	0.013	0	0.1	**0**	0	0.1
20	5	0.029	0.000	0.6	0.027	0.008	0.7	0.027	0.008	0.8	0.027	0.008	0.8	0.026	0	0.9	0.026	0	0.9
20	10	0.079	0.038	2.7	**0.048**	0.038	3.1	**0.048**	0.038	3.3	0.052	0.038	3.4	**0.049**	0.038	3.5	**0.047**	0.038	3.7
20	15	0.041	0	2.9	**0.003**	0	3.4	0.005	0	3.6	0.006	0	3.7	**0.004**	0	3.8	**0.003**	0	4.1
20	20	0.044	0	3.2	**0.008**	0	3.7	**0.004**	0	3.9	**0.007**	0	4.0	**0.004**	0	4.1	**0.004**	0	4.3
30	5	0.040	0	1.1	0.025	0.012	1.2	0.021	0.012	1.3	0.021	0.012	1.3	0.019	0	1.4	**0.012**	0	1.5
30	10	0.239	0.071	9.3	0.202	0	11.3	0.212	0.005	12.2	0.249	0.029	12.7	0.209	0.005	12.2	**0.171**	0	12.7
30	15	0.149	0.021	10.9	**0.073**	-0.004	12.9	**0.082**	-0.004	14.0	0.104	-0.004	14.7	**0.075**	-0.004	13.9	**0.073**	-0.004	14.3
30	20	0.153	0	13.2	**0.096**	0	15.3	**0.093**	0	16.7	**0.095**	0	17.6	**0.093**	0	16.5	**0.093**	0	17.0
40	5	0.026	0	1.7	**0.022**	0	1.9	**0.018**	0	2.0	**0.021**	0	2.2	**0.017**	0	2.0	**0.017**	0	2.2
40	10	0.380	0.102	9.2	0.371	0.117	11.2	0.416	0.166	12.3	0.460	0.202	12.8	0.378	0.094	12.0	**0.348**	0.053	12.4
40	15	**0.279**	-0.016	21.3	**0.285**	0.000	25.9	0.342	0.034	28.7	0.396	0.035	30.2	**0.274**	-0.036	27.2	**0.271**	-0.041	27.2
40	20	0.231	-0.057	24.3	**0.158**	-0.069	29.3	**0.165**	-0.078	32.5	0.212	-0.060	34.4	**0.162**	-0.060	30.7	**0.158**	-0.087	30.9
50	5	0	0	0.5	0	0	0.5	0	0	0.5	0	0	0.5	0	0	0.5	0	0	0.6
50	10	0.426	0.157	13.6	0.432	0.171	16.8	0.439	0.194	18.5	0.466	0.214	19.1	0.407	0.157	17.0	**0.349**	0.144	17.7
50	15	0.491	0.148	29.0	0.562	0.208	35.7	0.641	0.253	39.7	0.763	0.370	41.1	0.485	0.181	36.8	**0.450**	0.143	36.1
50	20	**0.244**	-0.122	37.3	**0.250**	-0.075	45.2	0.304	-0.080	50.3	0.410	0.035	52.3	0.269	-0.034	46.9	**0.242**	-0.069	45.4

To be continued ...

Table 17 (continued)

Instance set		Algorithms																
		$IILS_1$			$IILS_2$			$IILS_3$			$IILS_4$			RILS			QILS		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
60	5	0	0	0.7	0	0	0.8	0	0	0.8	0	0	0.9	0	0	0.8	0	0	0.9
60	10	0.319	0.121	20.4	0.326	0.178	24.9	0.354	0.233	28.1	0.400	0.220	29.7	0.290	0.133	26.1	**0.260**	0.062	26.8
60	15	0.405	0.122	35.7	0.441	0.137	43.3	0.536	0.163	48.0	0.609	0.295	50.7	0.400	0.119	43.7	**0.369**	0.088	42.4
60	20	0.329	-0.023	47.4	0.344	0.015	57.0	0.419	-0.012	63.2	0.546	0.132	67.5	0.326	-0.035	57.2	**0.291**	-0.120	54.7
Average		0.167	0.023	11.9	0.154	0.032	14.3	0.172	0.039	15.9	0.202	0.064	16.7	0.146	0.023	14.9	**0.133**	0.009	14.8

Table 18 – Performance comparison of QILS, RILS and $IILS_d$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.001, 0.037)$.

Instance set		Algorithms																
		$IILS_1$			$IILS_2$			$IILS_3$			$IILS_4$			RILS			QILS		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
100	20	0.250	-0.007	110	0.317	0.045	135	0.409	0.133	153	0.549	0.290	161	0.242	-0.034	146	**0.142**	-0.107	134
100	40	0.227	-0.064	147	0.215	-0.121	172	0.366	0.053	190	0.506	0.193	203	**0.208**	-0.092	184	**0.189**	-0.156	171
100	60	0.218	-0.104	154	0.207	-0.095	180	0.276	-0.005	223	0.450	0.196	230	0.215	-0.038	185	**0.177**	-0.111	177
200	20	0.443	0.212	243	0.430	0.199	303	0.442	0.255	343	0.504	0.322	372	0.238	0.011	306	**0.123**	-0.120	281
200	40	-0.071	-0.280	586	-0.015	-0.270	726	0.095	-0.199	834	0.201	-0.039	908	-0.074	-0.323	763	**-0.143**	-0.450	667
200	60	-0.137	-0.366	855	-0.114	-0.381	1046	0.040	-0.169	1194	0.141	-0.104	1298	-0.116	-0.369	1067	**-0.159**	-0.397	975
300	20	0.286	0.112	425	0.273	0.096	529	0.284	0.127	621	0.287	0.102	653	0.088	-0.081	553	**-0.020**	-0.172	492
300	40	0.025	-0.211	931	0.095	-0.160	1187	0.130	-0.116	1354	0.205	-0.035	1432	0.010	-0.297	1188	**-0.081**	-0.343	1081
300	60	0.005	-0.222	988	0.013	-0.237	1246	0.105	-0.130	1387	0.135	-0.163	1534	-0.001	-0.263	1269	**-0.068**	-0.367	1137
400	20	0.289	0.149	614	0.268	0.128	750	0.260	0.135	763	0.272	0.153	804	0.107	-0.021	680	**0.015**	-0.099	597
400	40	0.099	-0.155	1068	0.133	-0.078	1393	0.191	-0.024	1593	0.171	-0.032	1777	0.017	-0.226	1392	**-0.084**	-0.330	1263
400	60	-0.085	-0.290	1598	-0.060	-0.311	2061	0.008	-0.216	2435	0.052	-0.149	2712	-0.098	-0.378	2065	**-0.132**	-0.380	1924
500	20	0.242	0.110	761	0.224	0.085	936	0.221	0.099	1004	0.242	0.122	1034	0.082	-0.018	848	**0.025**	-0.074	731
500	40	0.061	-0.148	1434	0.119	-0.087	1924	0.139	-0.085	2211	0.154	-0.064	2357	-0.112	-0.298	1879	**-0.210**	-0.442	1759
500	60	-0.025	-0.195	2478	0.012	-0.198	3185	0.025	-0.165	3521	0.062	-0.145	3567	-0.036	-0.249	3199	**-0.061**	-0.275	2956
600	20	0.153	0.047	1019	0.150	0.042	1195	0.146	0.058	1251	0.155	0.043	1262	0.036	-0.058	1079	**-0.034**	-0.112	962
600	40	0.171	-0.009	1747	0.162	-0.015	2356	0.183	0.002	2736	0.196	0.035	2836	-0.137	-0.351	2258	**-0.229**	-0.415	2091
600	60	-0.056	-0.257	2169	-0.015	-0.207	2870	0.006	-0.198	3363	0.024	-0.178	3772	-0.086	-0.303	2892	**-0.182**	-0.376	2644
700	20	0.181	0.095	1394	0.190	0.079	1546	0.164	0.058	1564	0.172	0.070	1567	0.053	-0.037	1368	**-0.002**	-0.082	1225
700	40	0.055	-0.097	1965	0.103	-0.041	2688	0.124	-0.068	3153	0.142	-0.016	3284	-0.269	-0.430	2504	**-0.350**	-0.529	2326
700	60	-0.030	-0.222	2802	-0.021	-0.177	3763	0.019	-0.197	4512	0.039	-0.140	4896	-0.091	-0.267	3560	**-0.196**	-0.356	3285
800	20	0.148	0.061	1214	0.126	0.048	1380	0.112	0.050	1472	0.131	0.054	1477	0.042	-0.047	1317	**0.015**	-0.051	1186
800	40	0.094	-0.055	2747	0.087	-0.076	3632	0.116	-0.050	3731	0.156	0.008	3730	-0.267	-0.431	3489	**-0.373**	-0.568	3295
800	60	0.063	-0.128	3747	0.105	-0.118	5098	0.106	-0.061	5412	0.146	-0.004	5588	-0.067	-0.235	4864	**-0.127**	-0.314	4567
Average		0.109	-0.084	1300	0.125	-0.077	1679	0.165	-0.030	1876	0.212	0.022	1977	-0.001	-0.201	1627	**-0.073**	-0.276	1497

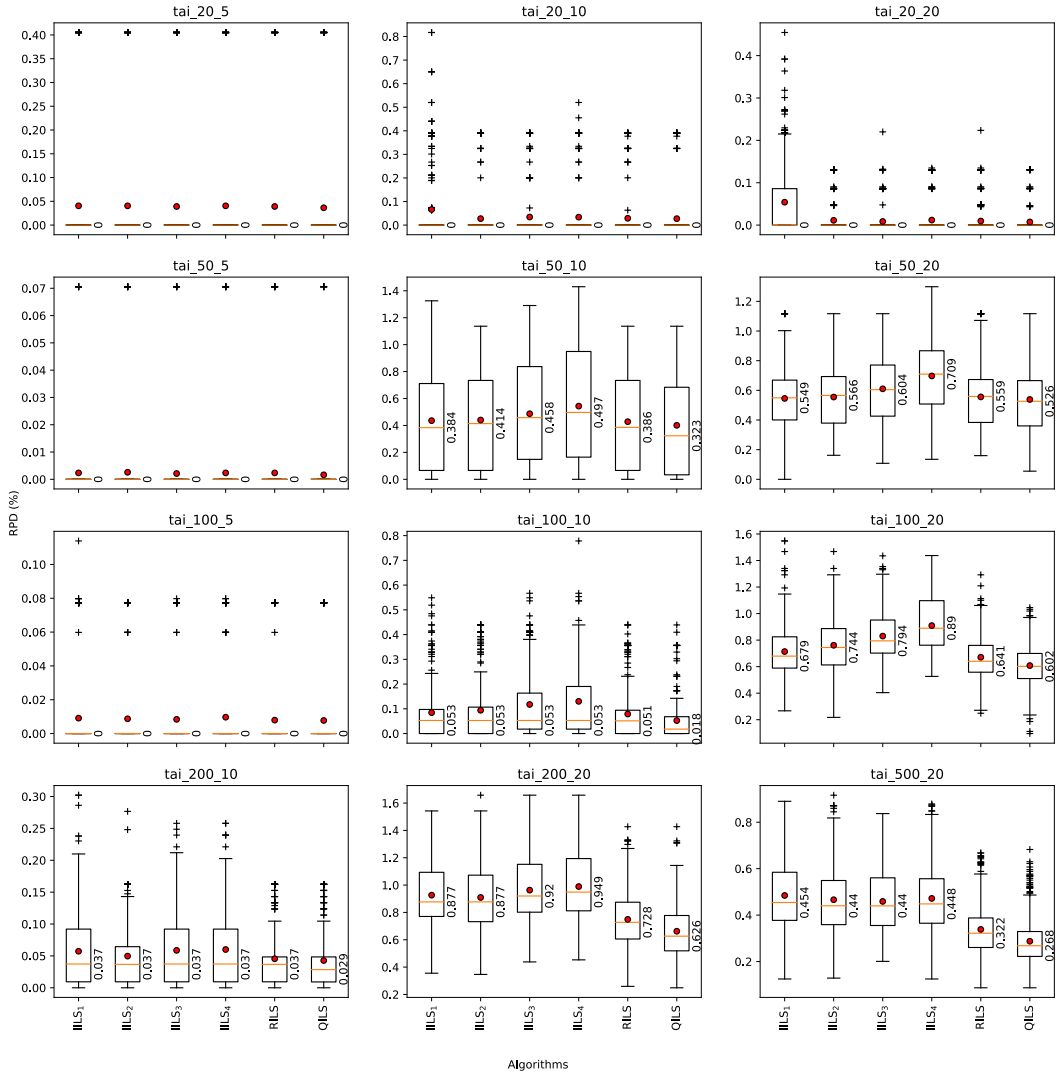

Figure 13 – Boxplot of QILS, RILS and IILS$_d$ based on RPD (%) for each instance set of *Taillard* dataset. The average performance of each algorithm is shown using red circles, and the median value is shown on the right side of each boxplot.

Comparison based on CPU time – Based on the CPU times reported in Tables 16 to 18, it can be seen that although QILS comes at the cost of a small complexity overhead compared to the ILS (see Section 4.5.3), it significantly improves the quality of solutions.

For smaller instance sets, QILS requires more CPU time, though this is not a strong limitation considering the short time scale. However, for large-sized instance sets, QILS experimentally requires lower CPU time under a fixed number of iterations than other algorithms except IILS$_1$, though still achieves significantly better solutions compared to IILS$_1$. Noteworthy, QILS outperforms all other algorithms in terms of both optimality gap and CPU time for *tai*_500_20 instance set and some other large-sized instances in *VRF-hard-large* dataset. In this regard, the boxplots of Figure 14 provide an overview of CPU times spent by different algorithms for 30 independent runs over *Taillard* dataset. They confirm a CPU time overhead for small instances, as well as better statistics for larger instance sets compared to RILS and IILS$_d$. The same behavior has also been observed for other two datasets in terms of CPU time.

The reason behind the better performance of QILS in terms of CPU time for larger

instance sets lies in the appropriateness of selecting perturbation operators. Indeed, the adjustment of the degree of perturbation in the proposed QILS framework keeps the local search from doing redundant efforts, including either finding the same local optimum as the recently visited one or repairing a highly perturbed solution to find a new local optimum. The former happens when the employed perturbation operator has been unable to bring out the solution from the local optimum, and the latter happens when a severe unnecessary perturbation has occurred. Experiments supporting this claim are provided in Section 4.5.2.3.

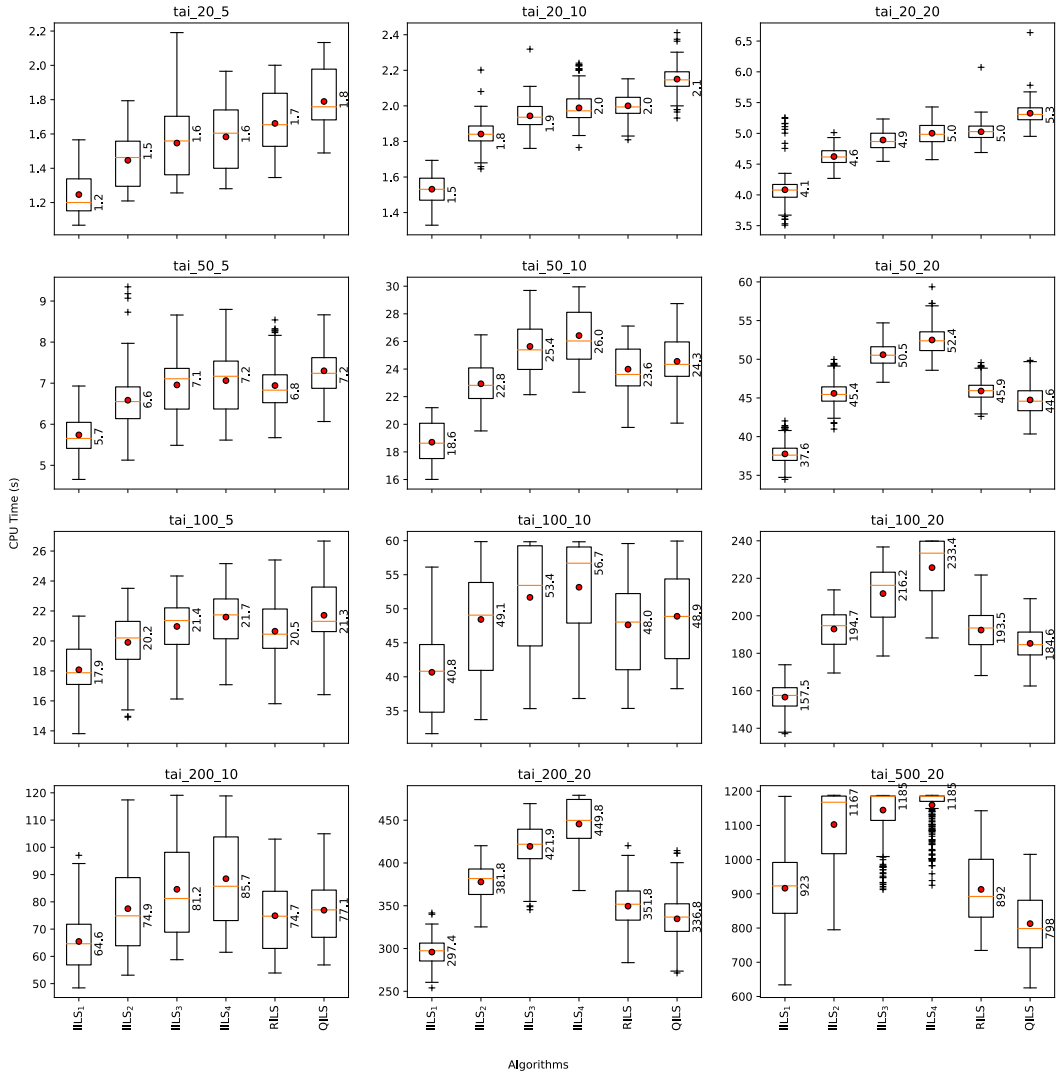

Figure 14 – Boxplot of QILS, RILS and IILS$_d$ based on CPU time (s) for each instance set of *Taillard* dataset

Comparison based on convergence behavior – In order to illustrate the performance of the algorithms throughout the search process, Figure 15 shows the convergence behavior of different algorithms for *Taillard* dataset. The number of iterations used as stopping criteria, as determined in Section 4.4.2, are shown in parentheses for each instance set. As it can be seen, the convergence curve of QILS always stays below the convergence curve of other algorithms throughout the search process, indicating a better exploration of the search space. This is in particular the case for large-sized instance sets, wherein QILS converges faster to the best found solution among all algorithms. It can be also seen that as long as the search proceeds, the convergence behavior of the

algorithms remains unchanged, as no crossings occur until the end of the search process. Accordingly, stopping the algorithms after extra iterations should not change the performance order of the algorithms.

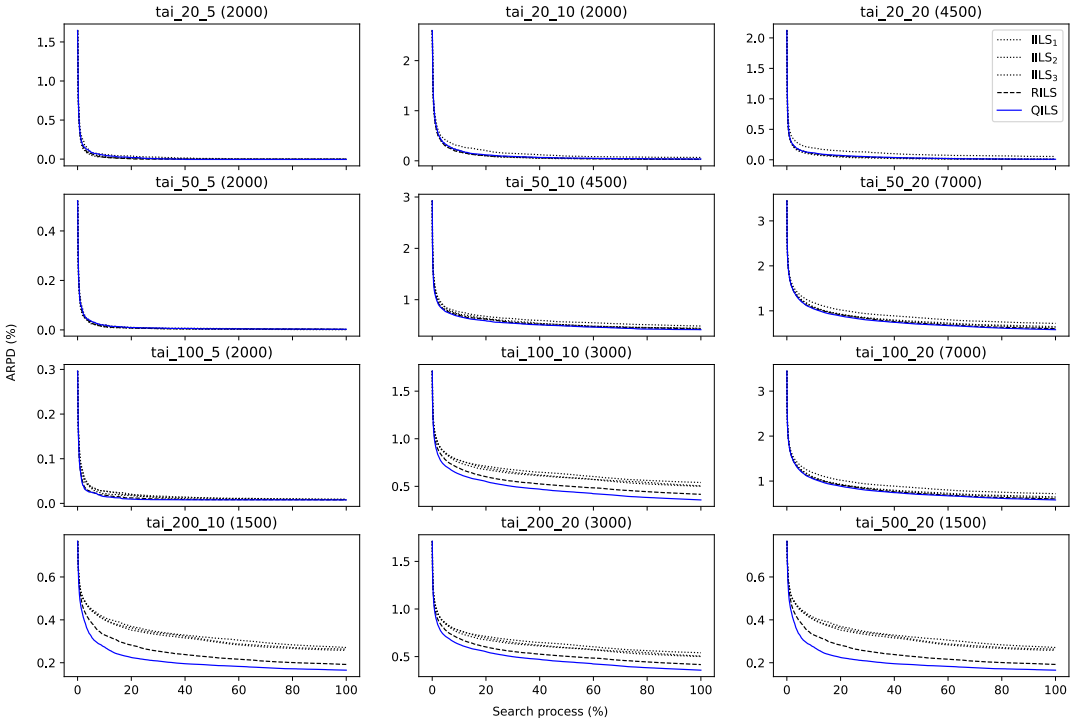

Figure 15 – Convergence rate of QILS, RILS and IILS$_d$ for each instance set of *Taillard* dataset

Overall comparison of algorithms – Table 19 summarizes the results of phase 1 and also shows the ARPD (%) of algorithms over all datasets. The QILS improves the performance of four IILS$_d$ in terms of optimality gaps by 50%, 49%, 56%, and 62%, respectively. Moreover, this improvement has been of 28% for RILS, which shows the contribution of a Q-learning based operator selection mechanism in an ILS algorithm with multiple perturbation operators. These improvements have been shown to be statistically significant.

Table 19 – ARPD of QILS, RILS, and IILS$_d$ for different datasets

Dataset	Algorithms					
	IILS$_1$	IILS$_2$	IILS$_3$	IILS$_4$	RILS	QILS
Taillard	0.285	0.281	0.302	0.325	0.246	**0.223**
VRF-hard-small	0.167	0.154	0.172	0.202	0.146	**0.133**
VRF-hard-large	0.109	0.125	0.165	0.212	-0.001	**-0.073**
Average	0.187	0.187	0.213	0.246	0.130	**0.094**

The overall robustness and computational effort of algorithms are illustrated in Figures 16 and 17 in terms of objective function value and CPU time, respectively. To compare the performance of the algorithms in terms of objective function value and CPU time across all instance sets and every execution, we normalize the corresponding values in a common range of $[0, 1]$ using $x^N = (x - x^{min})/(x^{max} - x^{min})$, where x^N and x are the normalized and real values, respectively. In addition, x^{min} and x^{max} are the minimum and maximum values over all instances, respectively. This normalization is performed separately for each dataset. Generally, in comparing the algorithms, lower values indicate better performance, while the performance gets worse as the values increase to one.

Looking at each dataset in Figure 16 and particularly *Taillard* and *VRF-hard-large* datasets, QILS yields better overall performance comparing to other algorithms with significantly lower median and mean values. Furthermore, as it can be seen in Figure 17, QILS in overall does not impose significant computational overhead when solving PFSP instances.

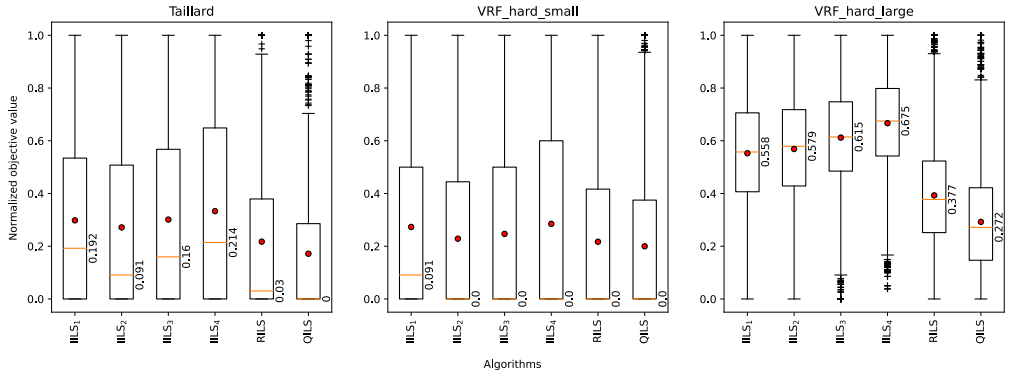

Figure 16 – Boxplot of QILS, RILS and IILS$_d$ based on normalized objective function value over all instances of each dataset

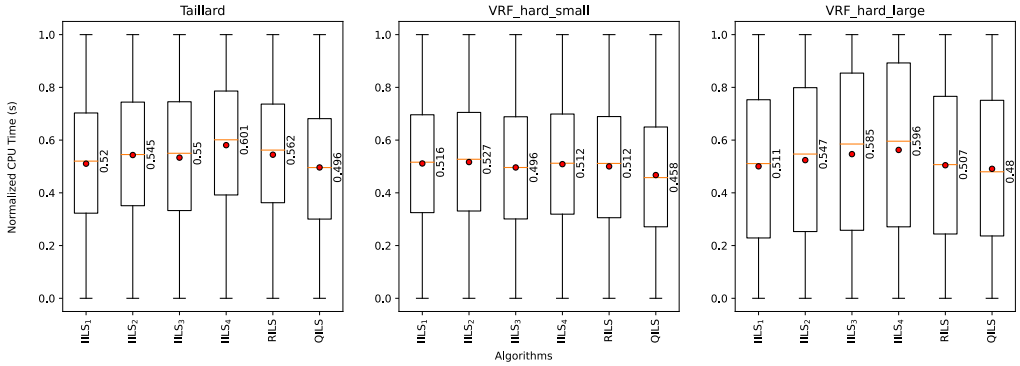

Figure 17 – Boxplot of QILS, RILS and IILS$_d$ based on normalized CPU time (s) over all instances of each dataset

At this step of the numerical experiments, we are able to answer the two first research questions. Regarding the first research question, employing multiple perturbation operators improves the performance of the ILS algorithm in finding better solutions. Moreover, regarding the second research question, incorporating the information about the status of the search into the selection of perturbation operators using Q-learning provides more efficient exploration of the search space and results in finding better solutions even with less computational effort for large-sized instances. Both of these results have been statistically verified.

4.5.2.2 Phase 2: comparison of QILS with state-of-the-art algorithms

In this phase, we compare the performance of QILS against seven well-known and efficient state-of-the-art algorithms and RILS to show how competitive QILS is comparing to the literature.

Comparison based on optimality gap – In this section, we analyze the performance of algorithms based on their optimality gaps. First, we start our analysis on *Taillard* dataset. Considering Tables 20 to 22 which respectively show the performance of QILS and the benchmark algorithms on *Taillard* dataset under three scales of $t \in \{60, 90, 120\}$,

it can be seen that QILS obtains better results than the benchmark algorithms in terms of both AvS and BS measures. The good performance of QILS can be attributed to its high exploration ability obtained by adding multiple perturbation operators and an adaptive selection of these operators according to the status of the search process as well as the history of their performance. The outperformance of QILS is also statistically verified for most of the instance sets. For some instance sets including "*tai*_20_5", "*tai*_20_10", and "*tai*_50_5", even though QILS obtains better results, the difference between QILS and other benchmark algorithms is not statistically significant. The reason is that these instance sets are among the easy PFSP instances and all algorithms are able to find solutions very close to the optimal solutions with small gaps.

To investigate the comparative performance of QILS on harder benchmark instances, we have extended our experiments also to two other datasets. Tables 23 to 25 and Tables 26 to 28 present the comparisons of QILS and benchmark algorithms on *VRF-hard-small* and *VRF-hard-large* datasets, respectively. For most of the instance sets in *VRF-hard-small* dataset, all algorithms have been able to achieve optimal solutions or solutions with small gaps. However, QILS still demonstrates higher performance compared to other benchmark algorithms. Considering the results on *VRF-hard-small* dataset, as the size of instances increases, we clearly observe better performance of QILS.

Looking at the results on *VRF-hard-large* dataset, the outperformance of QILS becomes more significant in finding good solutions for such large instances. This finding is promising, since solving large PFSP instances could be challenging for any algorithm. Furthermore, looking at Table 28 with scale $t = 120$ reveals that QILS significantly improves the best-known solutions of the *VRF-hard-large* dataset by achieving negative gaps.

To illustrate the robustness of QILS in comparison with the benchmark algorithms under limited CPU time, Figure 18 depicts the boxplots of RPD (%) of the algorithms for *Taillard* dataset for scale $t = 120$. As it can be seen for almost all the instance sets, QILS achieves a set of solutions with lower median and mean values as well as lower standard deviations. The latter implies that QILS exhibits a more robust behavior comparing to the other algorithms, particularly in larger instance sets.

Overall comparison of algorithms – Finally, we compare the overall performance of QILS against benchmark algorithms over all datasets. In this regard, Table 29 summarizes the results of phase 2 and the last row shows the ARPD (%) of each algorithm over all datasets. Accordingly, the QILS improves the performance of benchmark algorithms IG_{DPS}, IG_{KTPG}, IG_{PS}, and IG_{FF^*} by 41%, 33%, 27%, and 29% in terms of optimality gap, respectively. Moreover, Figure 19 illustrates the overall robustness of algorithms for each dataset at scale $t = 120$ based on the normalized objective function values. The same technique of normalization has been used as the one in phase 1. Looking at each dataset and particularly *Taillard* and *VRF-hard-large* datasets, QILS yields better overall performance with lower median and mean values as well as lower standard deviations.

Table 20 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 60$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.004, 0.047)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
20	5	0.036	0	0.035	0	0.034	0	0.036	0	0.040	0	0.037	0	0.034	0	0.039	0	0.034	0
20	10	0.011	0	0.015	0	0.015	0	0.010	0	0.018	0	0.016	0	0.009	0	0.014	0	0.010	0
20	20	0.023	0	0.023	0	0.020	0	0.003	0	0.008	0	0.014	0.013	0.005	0	0.003	0	**0**	0
50	5	0.002	0	0.002	0	0.002	0	0.002	0	0.002	0	0.002	0	0.002	0	0.002	0	0.002	0
50	10	0.470	0.290	0.481	0.276	0.474	0.276	0.505	0.310	0.482	0.290	0.448	0.279	0.475	0.310	0.472	0.297	**0.446**	0.255
50	20	0.667	0.294	0.682	0.302	0.654	0.326	0.617	0.317	0.612	0.310	0.605	0.305	0.593	0.295	0.614	0.344	0.599	0.263
100	5	0.009	0.008	0.015	0.008	0.011	0.008	0.010	0.008	0.010	0.008	0.010	0.008	**0.008**	0.008	**0.008**	0.008	**0.008**	0.008
100	10	0.164	0.025	0.152	0.023	0.140	0.042	0.138	0.034	0.129	0.042	0.115	0.026	0.122	0.034	0.115	0.021	**0.087**	0.016
100	20	0.992	0.554	1.049	0.598	1.012	0.610	0.996	0.632	0.894	0.573	**0.860**	0.608	0.883	0.593	0.902	0.550	**0.837**	0.483
200	10	0.082	0.033	0.087	0.032	0.079	0.038	0.058	0.034	0.057	0.032	0.056	0.030	0.055	0.033	0.047	0.033	**0.044**	0.025
200	20	1.249	0.880	1.199	0.820	1.135	0.766	1.144	0.819	1.084	0.766	0.943	0.729	0.978	0.728	0.944	0.663	**0.846**	0.572
500	20	0.693	0.514	0.657	0.464	0.619	0.428	0.595	0.428	0.551	0.397	0.499	0.375	0.496	0.364	0.437	0.283	**0.350**	0.229
Average		0.599	0.329	0.716	0.452	0.588	0.333	0.343	0.215	0.324	0.201	0.3	0.198	0.305	0.197	0.3	0.183	**0.272**	0.154

Table 21 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 90$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.003, 0.019)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
20	5	0.032	0	0.034	0	0.032	0	0.034	0	0.036	0	0.034	0	0.033	0	0.035	0	0.028	0
20	10	0.008	0	0.006	0	0.005	0	**0.005**	0	**0.008**	0	0.011	0	**0.005**	0	**0.008**	0	**0.005**	0
20	20	0.016	0	0.016	0	0.016	0	**0.001**	0	0.006	0	0.010	0.009	0.003	0	**0**	0	**0**	0
50	5	0	0	0	0	0	0	0	0	0	0	0.001	0	0	0	0	0	0	0
50	10	0.429	0.273	0.431	0.276	0.424	0.269	0.464	0.289	0.431	0.276	**0.409**	0.276	0.439	0.282	0.436	0.279	**0.403**	0.255
50	20	0.589	0.264	0.612	0.262	0.576	0.256	0.562	0.242	0.543	0.250	0.544	0.259	0.534	0.244	0.557	0.307	0.539	0.193
100	5	**0.008**	0.008	0.014	0.008	0.011	0.008	0.009	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008
100	10	0.145	0.025	0.130	0.023	0.119	0.025	0.108	0.027	0.111	0.025	0.100	0.025	0.097	0.023	0.093	0.021	**0.059**	0.016
100	20	0.910	0.508	0.970	0.549	0.916	0.515	0.914	0.553	0.800	0.500	0.811	0.535	0.820	0.530	0.808	0.486	**0.754**	0.430
200	10	0.072	0.033	0.070	0.030	0.064	0.037	0.052	0.033	0.051	0.031	0.054	0.029	0.052	0.033	0.044	0.033	**0.042**	0.024
200	20	1.194	0.847	1.145	0.783	1.085	0.746	1.094	0.792	1.044	0.746	0.898	0.687	0.954	0.706	0.882	0.626	**0.785**	0.513
500	20	0.677	0.509	0.637	0.448	0.599	0.416	0.578	0.411	0.542	0.396	0.473	0.342	0.476	0.357	0.406	0.277	**0.325**	0.223
Average		0.34	0.206	0.339	0.198	0.321	0.189	0.318	0.196	0.299	0.186	0.279	0.181	0.285	0.182	0.273	0.17	**0.246**	0.138

Table 22 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 120$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.035)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
20	5	0.030	0	0.032	0	0.030	0	0.034	0	0.035	0	0.032	0	0.032	0	0.032	0	0.027	0
20	10	0.003	0	0.003	0	0.004	0.000	**0.003**	0	**0.003**	0	0.009	0	**0.002**	0	**0.007**	0	**0.002**	0
20	20	0.012	0	0.014	0	0.011	0	**0**	0	0.003	0	0.009	0.009	0.002	0	**0**	0	**0**	0
50	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	10	0.401	0.269	0.406	0.273	0.394	0.265	0.444	0.276	0.406	0.273	0.392	0.276	0.420	0.282	0.405	0.273	**0.372**	0.255
50	20	0.535	0.240	0.554	0.229	0.537	0.242	0.514	0.217	0.510	0.237	0.505	0.216	0.488	0.215	0.516	0.235	0.494	0.193
100	5	**0.008**	0.008	0.013	0.008	0.009	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008
100	10	0.133	0.025	0.114	0.023	0.110	0.021	0.092	0.027	0.094	0.021	0.090	0.025	0.081	0.019	0.073	0.021	**0.048**	0.016
100	20	0.856	0.486	0.898	0.514	0.857	0.463	0.863	0.553	0.769	0.478	0.776	0.502	0.791	0.521	0.759	0.465	**0.703**	0.386
200	10	0.062	0.033	0.063	0.030	0.059	0.036	0.049	0.032	0.049	0.030	0.050	0.029	0.048	0.033	0.042	0.029	**0.040**	0.024
200	20	1.159	0.790	1.107	0.763	1.047	0.723	1.051	0.764	1.011	0.723	0.874	0.660	0.926	0.691	0.838	0.569	**0.738**	0.480
500	20	0.664	0.493	0.622	0.441	0.590	0.408	0.564	0.396	0.535	0.394	0.455	0.328	0.463	0.342	0.387	0.258	**0.307**	0.220
Average		0.322	0.195	0.319	0.19	0.304	0.18	0.302	0.189	0.285	0.18	0.267	0.171	0.271	0.176	0.256	0.155	**0.228**	0.132

Table 23 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 60$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.017, 0.048)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
10	5	0	0	0	0	0	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
10	10	0	0	0	0	0	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
10	15	0	0	0	0	0	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
10	20	0	0	0	0	0	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
20	5	0.019	0.008	0.016	0	0.018	0	0.016	0.008	0.016	0.000	0.018	0.000	0.015	0.000	0.016	0.000	0.016	0.000
20	10	0.061	0.038	0.066	0.038	0.055	0.038	0.040	0.038	0.057	0.038	0.050	0.038	0.046	0.038	**0.043**	0.038	0.040	0.038
20	15	0.006	0	0.007	0	0.007	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
20	20	0.004	0	0.003	0	0.002	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
30	5	0.024	0	0.032	0	0.021	0	0.014	0.000	0.017	0.000	0.015	0.000	**0.013**	0.000	0.015	0.000	0.010	0.000
30	10	0.259	0.021	0.267	0.031	**0.256**	0.026	0.224	0.049	0.245	0.031	0.231	0.026	0.221	0.035	0.236	0.024	0.216	0.014
30	15	0.184	0.025	0.172	0	0.170	0.021	0.076	0.008	0.074	0.021	0.077	0.017	0.075	0.008	0.076	-0.004	0.075	-0.004
30	20	0.184	0	0.199	0	0.176	0	**0.090**	0.000	**0.098**	0.000	**0.094**	0.021	**0.089**	0.000	**0.089**	0.000	**0.089**	0.000
40	5	0.023	0	0.026	0	0.025	0.009	**0.014**	0.000	**0.017**	0.000	**0.014**	0.000	**0.015**	0.000	**0.010**	0.000	**0.010**	0.000
40	10	0.382	0.065	0.395	0.090	0.378	0.061	0.377	0.109	0.380	0.065	0.380	0.065	**0.366**	0.069	0.377	0.090	**0.350**	0.053
40	15	**0.359**	0.039	0.387	0.042	**0.347**	0.007	0.333	0.060	0.347	0.007	0.349	0.042	0.343	0.039	0.341	0.029	0.332	-0.009
40	20	0.313	-0.017	0.328	-0.050	0.327	-0.042	0.165	-0.062	0.207	-0.017	0.200	-0.026	0.204	-0.017	0.201	-0.054	0.199	-0.066
50	5	0	0	0	0	0	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
50	10	0.449	0.179	0.491	0.208	0.445	0.218	0.424	0.207	0.421	0.172	0.411	0.165	0.416	0.186	0.415	0.174	**0.371**	0.144
50	15	0.592	0.202	0.616	0.196	0.587	0.187	0.605	0.238	0.594	0.223	0.579	0.196	0.592	0.196	0.586	0.240	**0.533**	0.187
50	20	0.502	0.005	0.528	0.067	0.483	0.020	0.330	-0.047	0.344	0.020	0.338	0.040	**0.334**	0.020	0.363	-0.019	**0.326**	-0.029
60	5	0	0	0	0	0	0	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
60	10	0.355	0.123	0.373	0.110	0.359	0.128	0.363	0.172	0.322	0.123	**0.312**	0.104	0.334	0.110	0.327	0.144	**0.298**	0.074

To be continued ...

Table 23 (continued)

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
60	15	0.538	0.145	0.554	0.169	0.534	0.140	0.504	0.176	0.504	0.169	**0.488**	0.161	0.496	0.148	0.492	0.140	**0.466**	0.137
60	20	0.558	0.081	0.618	0.143	0.503	0.039	0.441	-0.006	0.435	0.039	0.417	0.095	0.425	0.055	0.428	0.036	**0.392**	-0.047
Average		0.2	0.038	0.212	0.044	0.196	0.035	0.167	0.04	0.17	0.037	0.165	0.039	0.166	0.037	0.167	0.035	**0.155**	0.02

Table 24 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 90$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.011, 0.042)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
10	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	5	0.014	0	0.014	0	0.014	0	0.014	0.008	0.015	0	0.014	0	0.013	0	0.014	0	0.014	0
20	10	0.055	0.038	0.062	0.038	0.054	0.038	0.039	0.038	0.051	0.038	0.046	0.038	0.041	0.038	**0.039**	0.038	**0.038**	0.038
20	15	0.003	0	0.004	0	0.005	0	0	0	0	0	0	0	0	0	0	0	0	0
20	20	0.003	0	0.002	0	0.001	0	0	0	0	0	0	0	0	0	0	0	0	0
30	5	0.016	0	0.028	0	0.020	0	**0.012**	0	**0.012**	0	0.014	0	**0.009**	0	0.011	0	**0.007**	0
30	10	0.210	0.021	0.208	-0.005	0.211	0.005	0.198	0.015	0.208	0.031	**0.185**	0.005	**0.178**	0.005	0.198	0.005	**0.168**	-0.005
30	15	0.147	0	0.142	0	0.134	0.021	0.051	-0.013	0.056	0.021	0.053	-0.013	0.049	-0.021	0.052	-0.025	0.050	-0.025
30	20	0.144	0	0.165	0	0.152	0	0.065	0	0.081	0	**0.079**	0.004	**0.069**	0	**0.066**	0	**0.065**	0
40	5	0.021	0	0.025	0	0.023	0.009	0.012	0	0.016	0	0.012	0	0.012	0	0.009	0	0.009	0
40	10	0.329	0.041	0.348	0.045	0.327	0.041	0.328	0.081	**0.326**	0.041	**0.319**	0.041	**0.322**	0.033	0.337	0.069	**0.302**	0.033
40	15	**0.295**	0.028	0.325	0.011	**0.286**	-0.014	0.275	0.018	0.286	-0.014	0.272	0.028	0.280	0.028	0.275	-0.036	0.273	-0.041
40	20	0.262	-0.063	0.271	-0.059	0.272	-0.060	0.125	-0.084	0.171	-0.063	0.151	-0.072	0.158	-0.063	0.146	-0.084	0.144	-0.087
50	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	10	0.412	0.143	0.453	0.179	0.399	0.179	0.392	0.161	0.385	0.143	0.382	0.136	0.386	0.143	0.370	0.137	**0.331**	0.131
50	15	0.504	0.154	0.541	0.149	0.509	0.163	0.549	0.193	0.522	0.149	0.506	0.149	0.514	0.149	0.503	0.184	**0.465**	0.143
50	20	0.409	-0.029	0.440	0.001	0.392	-0.033	**0.248**	-0.077	0.259	-0.033	0.250	-0.050	0.249	-0.033	0.275	-0.034	**0.247**	-0.082
60	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60	10	0.337	0.083	0.339	0.099	0.328	0.103	0.335	0.161	0.291	0.123	0.272	0.093	0.307	0.099	0.291	0.133	**0.264**	0.068
60	15	0.473	0.098	0.478	0.122	0.473	0.089	0.434	0.129	0.430	0.122	0.428	0.122	0.418	0.098	0.411	0.127	**0.382**	0.088
60	20	0.455	-0.032	0.516	0.048	0.404	-0.018	0.339	-0.075	0.351	-0.018	0.332	-0.033	0.334	-0.032	0.340	-0.011	**0.298**	-0.118
Average		0.17	0.02	0.182	0.026	0.167	0.022	0.142	0.023	0.144	0.023	0.138	0.019	0.139	0.018	0.139	0.021	**0.127**	0.006

Table 25 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 120$ on *VRF-hard-small* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.008, 0.049)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
10	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	5	0.011	0	0.013	0	0.011	0	0.011	0	0.015	0	0.014	0	0.010	0	0.011	0	0.011	0
20	10	0.047	0	0.059	0	0.051	0	**0.037**	0.019	0.044	0	0.044	0.025	**0.040**	0.038	**0.038**	0.038	**0.036**	0
20	15	0.002	0	0.003	0	0.003	0	0	0	0	0	0	0	0	0	0	0	0	0
20	20	0.003	0	0.001	0	0.001	0	0	0	0	0	0	0	0	0	0	0	0	0
30	5	0.014	0	0.025	0	0.018	0	**0.011**	0	**0.011**	0	0.012	0	**0.007**	0	**0.010**	0	**0.007**	0
30	10	0.177	-0.005	0.178	-0.005	0.179	-0.005	0.169	0	0.171	0.031	**0.160**	-0.005	**0.154**	0	0.170	0.005	**0.146**	-0.005
30	15	0.123	0	0.124	0	0.121	-0.004	0.042	-0.025	0.048	-0.004	0.043	-0.025	0.039	-0.025	0.043	-0.025	0.042	-0.025
30	20	0.129	0	0.142	0	0.132	0	**0.051**	0	0.069	0	0.056	0	**0.054**	0	**0.050**	0	**0.050**	0
40	5	0.018	0	0.022	0	0.023	0.009	0.011	0	0.015	0	0.011	0	0.011	0	0.009	0	0.009	0
40	10	0.304	0.005	0.314	0.024	0.305	0.021	0.304	0.081	**0.302**	0.005	**0.295**	0.005	**0.290**	0.024	0.314	0.069	**0.277**	0.004
40	15	**0.260**	-0.030	0.275	-0.027	**0.241**	-0.028	0.239	0.015	0.241	-0.028	0.239	-0.030	0.246	-0.030	0.238	-0.042	0.238	-0.044
40	20	0.222	-0.085	0.234	-0.059	0.242	-0.060	0.093	-0.094	0.124	-0.085	0.117	-0.094	0.119	-0.085	**0.109**	-0.094	**0.106**	-0.099
50	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	10	0.394	0.133	0.428	0.169	0.386	0.175	0.375	0.154	0.367	0.140	0.363	0.126	0.367	0.133	0.340	0.131	**0.309**	0.127
50	15	0.439	0.120	0.471	0.122	0.456	0.122	0.504	0.181	0.467	0.122	0.450	0.128	0.457	0.122	0.453	0.154	**0.412**	0.116
50	20	0.352	-0.040	0.385	-0.021	0.337	-0.077	**0.198**	-0.093	0.214	-0.077	0.207	-0.071	0.202	-0.077	0.220	-0.096	**0.194**	-0.109
60	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60	10	0.316	0.065	0.325	0.099	0.311	0.103	0.317	0.161	0.267	0.065	0.254	0.093	0.297	0.099	0.274	0.113	**0.235**	0.051
60	15	0.424	0.041	0.417	0.044	0.422	0.037	0.393	0.081	0.377	0.044	0.381	0.044	0.376	0.041	0.371	0.052	**0.323**	0.036
60	20	0.395	-0.111	0.446	0.032	0.307	-0.042	0.281	-0.096	0.286	-0.042	0.278	-0.042	0.282	-0.111	0.279	-0.109	**0.239**	-0.122
Average		0.151	0.004	0.161	0.016	0.148	0.01	0.127	0.016	0.126	0.007	0.122	0.006	0.123	0.005	0.122	0.008	**0.110**	-0.003

Table 26 – Results of comparing QILS with RILS and algorithms for scale $t = 60$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.047)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
100	20	0.606	0.162	0.715	0.347	0.624	0.291	0.579	0.248	0.485	0.162	0.460	0.106	0.469	0.089	0.477	0.155	**0.380**	0.090
100	40	0.604	0.164	0.664	0.287	0.594	0.264	0.367	0.055	**0.305**	0.023	**0.334**	-0.006	**0.326**	0.006	0.342	0.015	**0.304**	-0.009
100	60	0.494	0.124	0.520	0.240	0.494	0.192	0.221	-0.051	0.211	-0.003	0.208	-0.092	**0.192**	-0.094	0.223	-0.027	**0.172**	-0.124
200	20	0.800	0.507	0.845	0.544	0.767	0.448	0.702	0.471	0.552	0.281	0.513	0.170	0.519	0.221	0.452	0.170	**0.299**	0.079
200	40	0.724	0.376	0.919	0.496	0.748	0.322	0.341	0.040	0.283	0.040	0.297	-0.056	0.277	-0.024	0.286	-0.051	**0.202**	-0.087
200	60	0.568	0.160	0.700	0.356	0.632	0.307	0.277	0.016	0.232	0.007	0.247	-0.048	0.234	-0.047	0.258	-0.016	**0.179**	-0.126
300	20	0.629	0.425	0.593	0.400	0.527	0.331	0.461	0.297	0.394	0.116	0.322	0.097	0.341	0.077	0.224	0.013	**0.144**	-0.055
300	40	0.793	0.431	1.080	0.680	0.868	0.458	0.415	0.125	0.323	0.146	0.295	0.034	0.344	0.010	0.306	0.027	**0.220**	-0.026
300	60	0.722	0.402	0.890	0.551	0.794	0.490	0.276	0.046	**0.228**	0.017	0.238	0.001	0.251	-0.004	0.235	-0.014	**0.202**	-0.055
400	20	0.524	0.340	0.503	0.331	0.458	0.305	0.419	0.277	0.341	0.177	0.268	0.109	0.291	0.106	0.217	0.077	**0.103**	-0.025

To be continued ...

Table 26 (continued)

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
400	40	0.763	0.455	0.984	0.654	0.828	0.544	0.355	0.157	0.232	0.063	0.238	-0.002	0.258	0.047	0.224	-0.012	**0.149**	-0.104
400	60	0.747	0.439	0.920	0.558	0.818	0.529	0.200	-0.084	**0.170**	-0.072	**0.171**	-0.110	**0.160**	-0.081	**0.165**	-0.150	**0.157**	-0.159
500	20	0.446	0.270	0.435	0.297	0.416	0.270	0.336	0.214	0.275	0.147	0.224	0.063	0.230	0.104	0.166	0.052	**0.088**	-0.019
500	40	0.661	0.391	0.857	0.574	0.657	0.438	0.271	0.074	0.199	-0.075	0.144	-0.106	0.131	-0.115	0.077	-0.126	**0.003**	-0.255
500	60	0.759	0.450	0.952	0.653	0.823	0.535	0.211	-0.001	0.210	-0.025	**0.200**	-0.009	**0.202**	-0.011	**0.204**	-0.023	**0.189**	-0.024
600	20	0.348	0.222	0.332	0.220	0.304	0.168	0.222	0.119	0.152	0.047	0.135	0.017	0.139	0.056	0.107	0.020	**0.025**	-0.055
600	40	0.640	0.382	0.849	0.566	0.589	0.392	0.269	0.067	0.143	-0.039	0.120	-0.110	0.130	-0.078	0.057	-0.159	**-0.032**	-0.257
600	60	0.647	0.382	0.840	0.599	0.602	0.294	0.094	-0.126	0.065	-0.141	0.044	-0.148	0.050	-0.127	0.036	-0.182	**-0.011**	-0.200
700	20	0.347	0.231	0.350	0.235	0.320	0.200	0.244	0.141	0.151	0.039	0.139	0.019	0.145	0.038	0.128	0.019	**0.061**	-0.024
700	40	0.545	0.321	0.711	0.476	0.437	0.209	0.186	0.021	0.018	-0.114	-0.003	-0.181	-0.009	-0.201	-0.081	-0.230	**-0.166**	-0.341
700	60	0.587	0.331	0.752	0.497	0.564	0.295	0.086	-0.084	0.069	-0.120	0.036	-0.170	0.054	-0.144	0.021	-0.148	**-0.042**	-0.204
800	20	0.274	0.193	0.261	0.185	0.254	0.130	0.186	0.089	0.129	0.033	0.101	0.028	0.113	0.037	0.077	-0.011	**0.041**	-0.029
800	40	0.521	0.332	0.673	0.446	0.413	0.207	0.180	0.028	0.058	-0.082	0.014	-0.183	0.027	-0.164	-0.05	-0.221	**-0.137**	-0.316
800	60	0.635	0.412	0.847	0.665	0.571	0.382	0.145	-0.003	0.126	-0.075	0.105	-0.082	0.110	-0.072	0.085	-0.099	**0.035**	-0.134
Average		0.599	0.329	0.716	0.452	0.588	0.333	0.293	0.089	0.223	0.023	0.202	-0.027	0.208	-0.015	0.176	-0.038	**0.107**	-0.102

Table 27 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 90$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.049)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF^*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
100	20	0.507	0.103	0.611	0.230	0.504	0.193	0.481	0.171	0.382	0.045	0.349	-0.011	0.357	0.025	0.389	0.071	**0.257**	-0.011
100	40	0.516	0.085	0.541	0.142	0.483	0.162	0.256	-0.028	0.219	-0.072	**0.216**	-0.102	**0.202**	-0.098	0.222	-0.088	**0.179**	-0.147
100	60	0.392	0.062	0.415	0.114	0.407	0.133	0.142	-0.099	0.123	-0.118	0.127	-0.149	0.100	-0.165	0.108	-0.150	**0.070**	-0.178
200	20	0.744	0.455	0.777	0.490	0.708	0.395	0.645	0.436	0.507	0.264	0.455	0.142	0.462	0.164	0.378	0.142	**0.231**	-0.008
200	40	0.621	0.309	0.805	0.426	0.641	0.209	0.236	-0.021	0.204	-0.021	0.194	-0.118	0.164	-0.103	0.180	-0.115	**0.057**	-0.246
200	60	0.460	0.069	0.619	0.263	0.528	0.207	0.149	-0.121	0.138	-0.137	0.124	-0.188	0.130	-0.141	0.118	-0.159	**0.069**	-0.217
300	20	0.601	0.410	0.562	0.373	0.497	0.297	0.436	0.267	0.378	0.100	0.282	0.041	0.296	0.050	0.191	-0.020	**0.082**	-0.093
300	40	0.717	0.382	1.004	0.616	0.803	0.411	0.334	0.092	0.256	0.068	0.227	-0.030	0.276	-0.021	0.221	-0.060	**0.119**	-0.155
300	60	0.644	0.323	0.792	0.474	0.663	0.398	0.179	-0.044	0.156	-0.073	0.147	-0.088	0.157	-0.116	0.135	-0.102	**0.098**	-0.137
400	20	0.503	0.328	0.481	0.286	0.434	0.293	0.396	0.261	0.313	0.130	0.241	0.095	0.268	0.092	0.190	0.060	**0.062**	-0.042
400	40	0.712	0.392	0.922	0.593	0.745	0.443	0.290	0.080	0.168	-0.032	0.177	-0.042	0.187	-0.012	0.163	-0.060	**0.071**	-0.187
400	60	0.678	0.380	0.857	0.464	0.717	0.441	0.121	-0.160	0.114	-0.163	**0.089**	-0.182	0.095	-0.185	**0.071**	-0.206	**0.059**	-0.256
500	20	0.431	0.264	0.414	0.272	0.397	0.258	0.325	0.199	0.267	0.129	0.188	0.031	0.210	0.047	0.139	0.030	**0.062**	-0.049
500	40	0.618	0.346	0.810	0.543	0.594	0.359	0.217	0.016	0.129	-0.111	0.093	-0.130	0.103	-0.118	0.051	-0.149	**-0.055**	-0.298
500	60	0.702	0.392	0.905	0.599	0.743	0.449	0.163	-0.041	0.154	-0.032	**0.150**	-0.033	**0.142**	-0.036	0.147	-0.057	**0.123**	-0.096
600	20	0.334	0.213	0.318	0.202	0.296	0.159	0.212	0.112	0.139	0.015	0.110	-0.015	0.117	0.022	0.088	-0.015	**0**	-0.092
600	40	0.596	0.345	0.815	0.553	0.548	0.355	0.237	0.042	0.119	-0.109	0.076	-0.174	0.082	-0.131	0.016	-0.211	**-0.091**	-0.322
600	60	0.596	0.355	0.801	0.561	0.533	0.228	0.047	-0.149	0.008	-0.192	-0.005	-0.244	-0.013	-0.229	-0.017	-0.238	**-0.060**	-0.244
700	20	0.339	0.226	0.336	0.221	0.302	0.184	0.237	0.139	0.132	0.007	0.121	-0.003	0.109	0.002	0.110	-0.003	**0.042**	-0.039
700	40	0.518	0.293	0.667	0.451	0.401	0.179	0.159	-0.009	-0.030	-0.193	-0.048	-0.245	-0.068	-0.243	-0.115	-0.276	**-0.210**	-0.381
700	60	0.551	0.285	0.722	0.468	0.513	0.239	0.051	-0.122	0.041	-0.161	0.003	-0.206	0.016	-0.180	-0.015	-0.186	**-0.086**	-0.247
800	20	0.263	0.184	0.249	0.178	0.240	0.126	0.179	0.083	0.104	0.010	0.086	-0.019	0.095	-0.004	0.066	-0.017	**0.030**	-0.040

To be continued ...

Table 27 (continued)

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
800	40	0.488	0.305	0.641	0.403	0.384	0.177	0.155	0.010	0.029	-0.150	-0.026	-0.246	0.000	-0.184	-0.082	-0.262	**-0.181**	-0.353
800	60	0.601	0.376	0.824	0.644	0.527	0.342	0.116	-0.062	0.091	-0.095	0.069	-0.104	0.077	-0.098	0.048	-0.120	**-0.001**	-0.163
Average		0.547	0.287	0.662	0.399	0.525	0.277	0.24	0.044	0.173	-0.037	0.144	-0.084	0.149	-0.069	0.117	-0.091	**0.039**	-0.167

Table 28 – Results of comparing QILS with RILS and benchmark algorithms for scale $t = 120$ on *VRF-hard-large* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.002, 0.049)$.

Inst.		Algorithms																	
set		IG_{RS}		IG_{PTL}		IG_{FF}		IG_{DPS}		IG_{KTPG}		IG_{PS}		IG_{FF*}		RILS		QILS	
n	m	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS	AvS	BS
100	20	0.428	0.027	0.537	0.177	0.434	0.092	0.412	0.139	0.309	-0.034	0.273	-0.052	0.284	-0.056	0.303	0.051	**0.180**	-0.089
100	40	0.434	0.014	0.456	0.114	0.397	0.046	0.158	-0.127	0.135	-0.168	0.121	-0.196	0.133	-0.174	0.154	-0.170	**0.100**	-0.251
100	60	0.324	0.036	0.360	0.057	0.343	0.027	0.062	-0.150	0.051	-0.154	0.036	-0.181	0.031	-0.226	0.038	-0.210	**0**	-0.262
200	20	0.710	0.419	0.728	0.465	0.670	0.373	0.605	0.370	0.425	0.212	0.403	0.087	0.410	0.143	0.334	0.087	**0.162**	-0.094
200	40	0.551	0.229	0.714	0.329	0.570	0.169	0.157	-0.088	0.146	-0.110	0.120	-0.149	0.101	-0.184	0.097	-0.181	**-0.061**	-0.375
200	60	0.377	0.005	0.545	0.196	0.452	0.133	0.069	-0.183	0.075	-0.193	0.038	-0.220	0.054	-0.267	0.025	-0.243	**-0.036**	-0.308
300	20	0.583	0.387	0.542	0.345	0.480	0.287	0.418	0.258	0.342	0.081	0.243	0.008	0.259	0.035	0.163	-0.032	**0.039**	-0.120
300	40	0.676	0.349	0.950	0.565	0.742	0.349	0.286	0.024	0.213	0.002	0.185	-0.081	0.220	-0.088	0.161	-0.131	**0.043**	-0.199
300	60	0.584	0.250	0.728	0.394	0.580	0.289	0.097	-0.123	0.088	-0.129	0.071	-0.166	0.087	-0.235	0.049	-0.234	**0.013**	-0.242
400	20	0.487	0.324	0.463	0.278	0.416	0.282	0.381	0.240	0.306	0.106	0.217	0.090	0.248	0.088	0.172	0.054	**0.037**	-0.076
400	40	0.671	0.353	0.885	0.535	0.683	0.379	0.253	0.044	0.123	-0.069	0.143	-0.085	0.162	-0.044	0.134	-0.090	**0.003**	-0.232
400	60	0.633	0.334	0.818	0.406	0.644	0.386	0.058	-0.197	0.060	-0.197	0.025	-0.240	0.036	-0.238	0.003	-0.280	**-0.019**	-0.310
500	20	0.421	0.257	0.402	0.263	0.388	0.257	0.319	0.196	0.252	0.080	0.174	0.020	0.189	0.028	0.123	0.020	**0.041**	-0.064
500	40	0.583	0.279	0.783	0.518	0.549	0.322	0.178	-0.024	0.091	-0.139	0.067	-0.160	0.079	-0.147	0.020	-0.163	**-0.111**	-0.320
500	60	0.657	0.361	0.864	0.572	0.684	0.391	0.124	-0.086	0.108	-0.054	0.097	-0.068	0.090	-0.078	0.093	-0.103	**0.071**	-0.133
600	20	0.324	0.212	0.304	0.193	0.283	0.153	0.205	0.104	0.120	-0.007	0.093	-0.030	0.103	-0.002	0.069	-0.030	**-0.015**	-0.098
600	40	0.563	0.317	0.790	0.535	0.513	0.328	0.209	0.020	0.091	-0.146	0.056	-0.202	0.070	-0.149	-0.011	-0.243	**-0.118**	-0.334
600	60	0.563	0.314	0.765	0.538	0.485	0.175	0.011	-0.164	-0.025	-0.223	-0.029	-0.264	-0.046	-0.262	-0.041	-0.260	**-0.095**	-0.282
700	20	0.330	0.210	0.322	0.216	0.297	0.182	0.231	0.136	0.122	-0.007	0.115	-0.007	0.094	-0.007	0.098	-0.007	**0.023**	-0.060
700	40	0.491	0.268	0.639	0.420	0.375	0.153	0.133	-0.031	-0.055	-0.260	-0.068	-0.280	-0.095	-0.276	-0.144	-0.299	**-0.242**	-0.401
700	60	0.520	0.254	0.696	0.443	0.484	0.216	0.021	-0.157	-0.003	-0.205	-0.038	-0.225	-0.018	-0.207	-0.043	-0.205	**-0.118**	-0.290
800	20	0.258	0.180	0.241	0.163	0.232	0.120	0.175	0.082	0.096	-0.002	0.081	-0.023	0.087	-0.011	0.057	-0.022	**0.024**	-0.045
800	40	0.464	0.287	0.615	0.394	0.357	0.158	0.137	-0.015	-0.004	-0.178	-0.043	-0.269	-0.019	-0.193	-0.105	-0.269	**-0.212**	-0.404
800	60	0.580	0.362	0.803	0.610	0.494	0.300	0.090	-0.083	0.077	-0.105	0.044	-0.117	0.049	-0.110	0.018	-0.138	**-0.026**	-0.186
Average		0.509	0.251	0.623	0.364	0.481	0.232	0.2	0.008	0.131	-0.079	0.101	-0.117	0.109	-0.111	0.074	-0.129	**-0.013**	-0.216

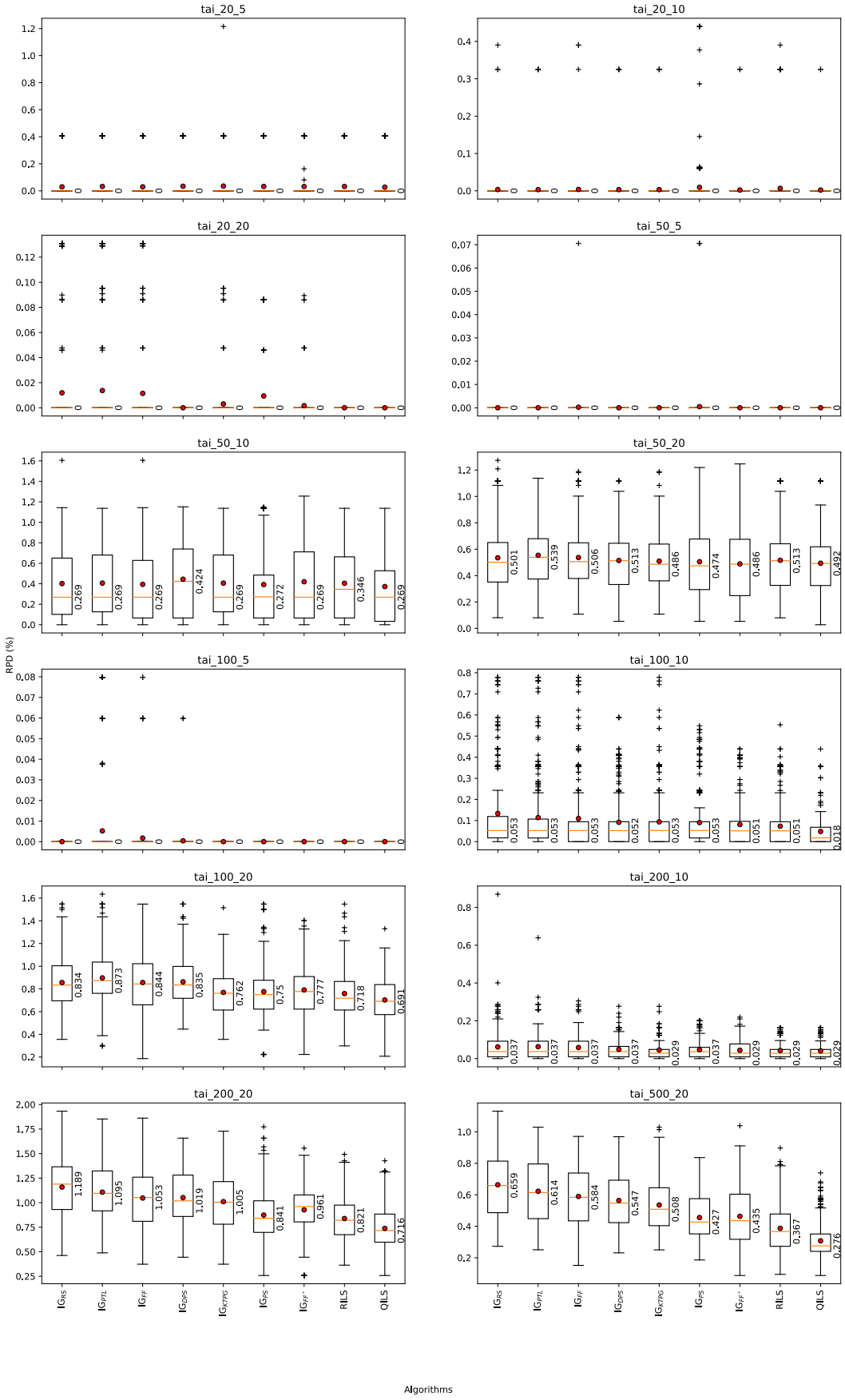

Figure 18 – Boxplot of QILS, RILS, and benchmark algorithms based on RPD (%) for scale $t = 120$ for each instance set of *Taillard* dataset

Table 29 – ARPD of QILS, RILS, and benchmark algorithms for different datasets

Dataset	Algorithm								
	IG_{RS}	IG_{PTL}	IG_{FF}	IG_{DPS}	IG_{KTPG}	IG_{PS}	IG_{FF*}	RIG	QIG
					$t = 60$				
Taillard	0.367	0.366	0.35	0.343	0.324	0.300	0.305	0.300	**0.272**
VRF-small	0.2	0.212	0.196	0.167	0.170	0.165	0.166	0.167	**0.155**
VRF-large	0.599	0.716	0.588	0.293	0.223	0.202	0.208	0.176	**0.107**
Average	0.389	0.431	0.378	0.268	0.239	0.222	0.226	0.214	**0.178**
					$t = 90$				
Taillard	0.34	0.339	0.321	0.318	0.299	0.279	0.285	0.273	**0.246**
VRF-small	0.17	0.182	0.167	0.142	0.144	0.138	0.139	0.139	**0.127**
VRF-large	0.547	0.662	0.525	0.240	0.173	0.144	0.149	0.117	**0.039**
Average	0.352	0.394	0.338	0.233	0.205	0.187	0.191	0.176	**0.137**
					$t = 120$				
Taillard	0.322	0.319	0.304	0.302	0.285	0.267	0.271	0.256	**0.228**
VRF-small	0.151	0.161	0.148	0.127	0.126	0.122	0.123	0.122	**0.110**
VRF-large	0.509	0.623	0.481	0.200	0.131	0.101	0.109	0.074	**-0.013**
Average	0.327	0.368	0.311	0.210	0.181	0.163	0.168	0.151	**0.108**

We come to the conclusion that not only visually, but also from a statistical viewpoint, the proposed QILS shows a significantly better performance. It answers the third research question, that the proposed QILS framework performs better than benchmark algorithms from the literature.

4.5.2.3 Adaptiveness of operators to the problem instances

In this section, we aim at answering the fourth research question and investigate how the proposed QILS framework automatically adapts the perturbation operators to the problem instance at hand, and determine if this adaptiveness has been significant among different operators. For this aim, we analyze how much each perturbation operator contributes to the total improvement of the initial solution and also illustrate how (where and at which frequency) different operators have been employed throughout the search process.

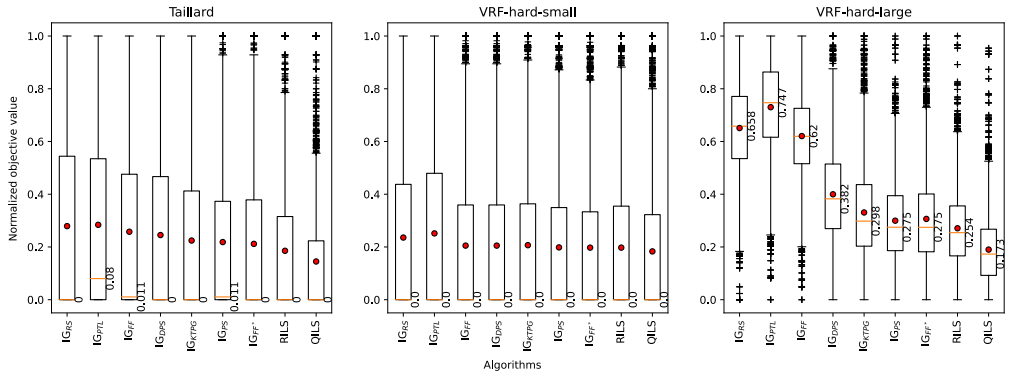

Figure 19 – Boxplot of QILS and benchmark IGs based on normalized objective value over all instances of each dataset

For each perturbation operator $a \in A$ (A is the set of actions defined in Section 4.3.2), we calculate the total gap improvement $GI(a)$ obtained in a number of $SI(a)$ successful applications. Note that a single application of the perturbation operator a is followed by a full neighborhood exploitation using the local search. The ratio $R(a) = GI(a)/SI(a)$ then indicates the average expected improvement that a successful exploration of operator a would produce. Accordingly, we compute the Relative Improvement index for each

perturbation operator a with respect to the set of all available perturbation operators as Equation (4.9).

$$RI(a) = \frac{R(a)}{\sum_{a' \in A} R(a')} \times 100 \tag{4.9}$$

In this regard, Figure 20 shows the Relative Improvement index of each perturbation operator $a \in A$ for sample instances from instance sets "tai_50_20", "tai_100_20", "tai_200_20", and "tai_500_20" of *Taillard* dataset. As it can be seen, all the perturbation operators significantly contribute to the overall improvement in each instance. However, these contributions differ from an instance set to another. Accordingly, it can be concluded that depending on the problem instance, the proposed QILS adapts the perturbation operators to the instance at hand throughout the search process.

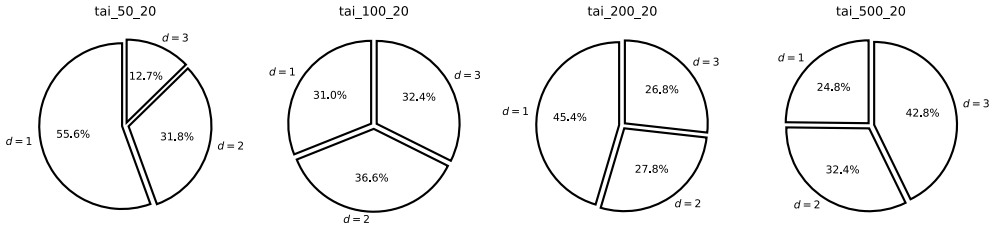

Figure 20 – Relative improvement index for each perturbation operator in certain instance sets of *Taillard* dataset

To show in more detail how the proposed QILS framework adapts different perturbation operators when solving different instances, Figure 21 illustrates the perturbation operator that has been selected and used at each iteration of the search process for the same instances of Figure 20. As it can be observed, QILS has selected different operators at different stages of the search process and all three perturbation operators (i.e., $d \in \{1, 2, 3\}$) have been effectively employed throughout the search process of each instance.

In instance "tai_50_20", one may see that the density of employing the perturbation operator $d = 3$ (with higher exploration ability) is higher than the two other operators at the beginning of the search process. Vice versa, the density of the perturbation operator $d = 1$ (with weaker exploration ability) is higher at the end of the search process. The reason can be attributed to a typical intuition in the literature [Tal09] that meta-heuristics put more efforts on exploration at the beginning of the search process to well explore unseen regions and less exploration (or higher exploitation) at the end of the search process to well exploit the promising found regions. This interpretation is in line with considering the selection of perturbation operators as a function of iterations.

This phenomenon happened in instance "tai_50_20" can be interpreted from a second viewpoint related to the quality of the initial solution. Since we use NEH heuristic, the generated initial solution of instance "tai_50_20" has already a good quality in terms of objective function (i.e., it is already a good local optimum). Therefore, a good initial solution forces the proposed algorithm to employ perturbation operators with higher strength to allow the solution to escape from local optima. In this situation, most effort is used to escape from these local optima rather than exploring unseen regions. This interpretation relates the selection of perturbation operators mostly to the status of the search and not as a function of iterations.

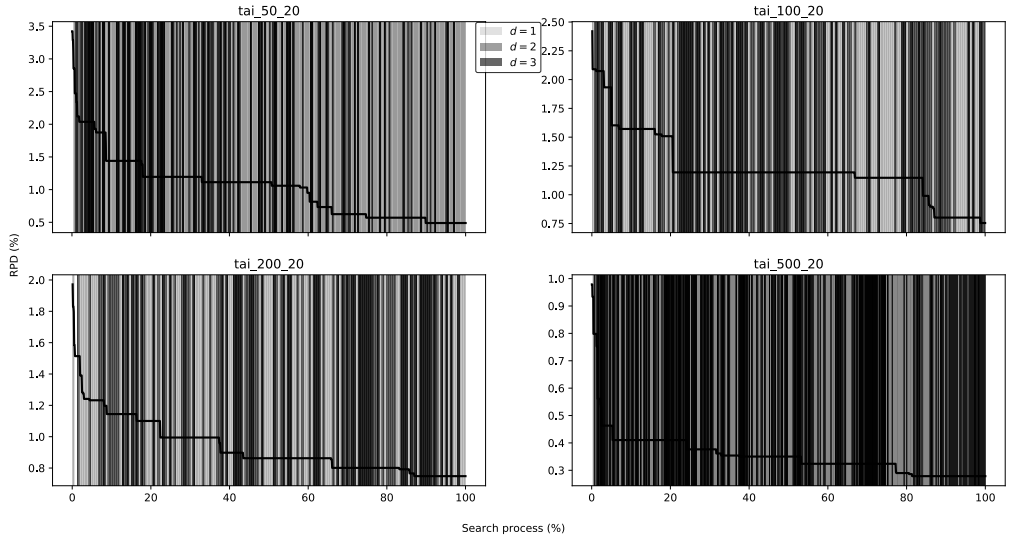

Figure 21 – Application of each perturbation operator at each step of the search process in certain instances of *Taillard* dataset

After looking at other three instances of Figure 21, and checking the majority of instances over all datasets, we observed that the first interpretation (i.e., selection as a function of iterations) is not the case in our problem. For example, in other instances of Figure 21, there is not a particular pattern on the employment of different perturbation operators at particular iterations. Accordingly, we conclude that the proposed QILS framework adapts the perturbation operators to each problem instance based on the status of the search and adjust the degree of exploration to guide the search process toward more promising solutions.

4.5.2.4 Sensitivity analysis

In this section, we perform a sensitivity analysis on the performance of the proposed QILS framework with regard to its parameters levels, the size of the action set (i.e., $|A|$), and the reward function.

QILS's parameters – As explained in Section 4.4.5, we tune the parameters of the proposed QILS framework using RSM that finds the optimum level of parameters. However, in this section, to illustrate how much the performance of the proposed QILS is sensitive to different levels of parameters, we do a sensitivity analysis on all its parameters including τ, ϵ, β, α, γ, E, and η on *Taillard* dataset.

The results of this analysis has been provided in Figure 22, where for each subplot, the corresponding parameter value has been changed while other parameters are kept fixed at their tuned optimum level as in Table 13. As it can be seen, the performance of the proposed QILS framework is sensitive to the level of parameters, and this sensitivity varies from one parameter to another. The lowest and the highest sensitivities belong to the parameters α (learning rate) and E (number of episodes), respectively. However, these sensitivities are not very severe to force the users to spend significant effort on the parameter tuning. As also mentioned above, the better quality of the solutions that the proposed QILS obtains prevails the parameter tuning effort.

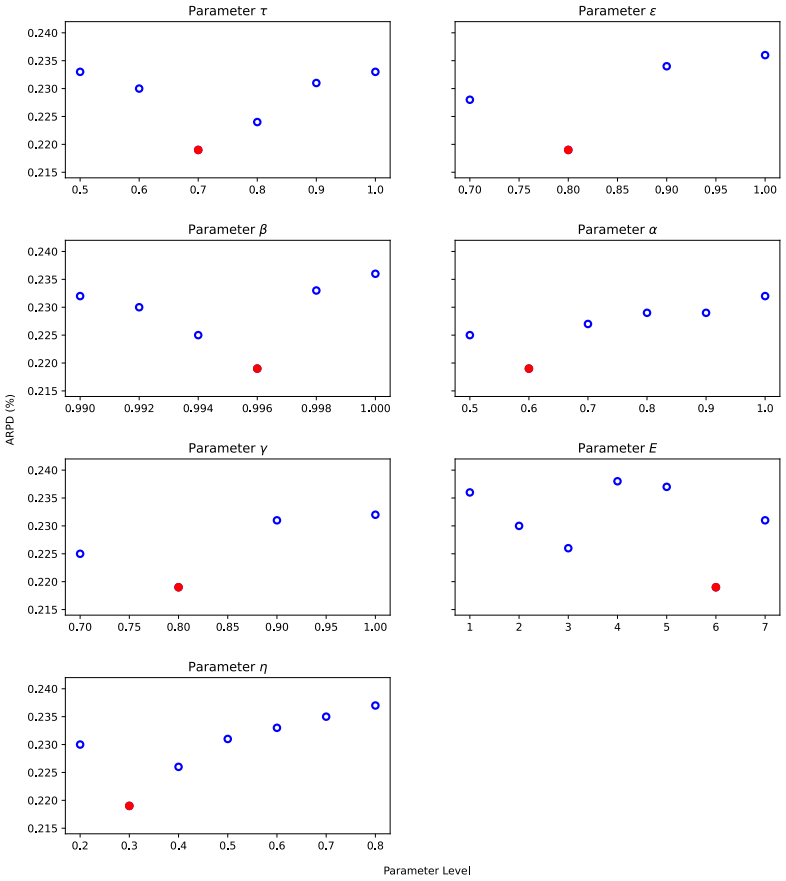

Figure 22 – Sensitivity of QILS to its parameters. Values found in Table 13 are highlighted in red.

Action set size $|A|$ – In this section, we perform a sensitivity analysis on the performance of the proposed QILS framework with regard to the size of the action set (i.e., $|A|$). For this aim, we vary $|A|$ from 2 (the lowest possible size of $|A|$ for considering multiple actions in QILS) until a maximum number of 10 actions. Considering that IILS$_d$ with perturbation operators of $d = 1$, $d = 2$, and $d = 3$ individually shows the best performance among other values of d (i.e., the performance of IILS$_d$ with higher values of $d \geq 4$ are statistically dominated by the values of $d \leq 3$), we incorporate these three values to construct different action lists in QILS. On the other hand, for the QILS with $|A| = 2$, we consider three different action lists as $A = \{1, 2\}$, $A = \{1, 3\}$, and $A = \{2, 3\}$. Furthermore, the QILS with $|A| = 3$ is formed based on $d = 1$, $d = 2$, and $d = 3$ (i.e., $A = \{1, 2, 3\}$). Moreover, we keep all $d = 1$, $d = 2$, and $d = 3$ and add $d = 4, 5, ..., 10$ one by one incrementally to form QILS with $4 \leq |A| \leq 10$. The reason is to keep the best actions (i.e., $d = 1$, $d = 2$, and $d = 3$) always in the list of available actions. Accordingly, we have seven more sets of actions as $A = \{1 - 4\}$, $A = \{1 - 5\}$, ..., and $A = \{1 - 10\}$ that correspond to action lists with $d = 1, 2, 3, 4$, $d = 1, 2, 3, 4, 5$, ..., and $d = 1, 2, ..., 10$, respectively. Finally, the QILS is executed with 11 sets of actions to investigate the sensitivity of QILS to the size of action set. In the meantime, the performance of QILS is also compared with its corresponding RILS.

Figure 23 shows the average performance of QILS and RILS for different sets of actions with different sizes on *Taillard* dataset. The detailed results of all executions for each instance set has been provided in Table 31. It can be seen that the QILS with $A = \{1 - 3\}$ provides the minimum ARPD and there is not visually a big difference between the performance of QILSs with different sizes of actions. In addition, in a statistical

viewpoint using Wilcoxon signed rank test, there is no significant difference between QILS with $A = \{1-3\}$ and all other QILSs, except the QILSs with $A = \{1-8\}, \{1-9\}, \{1-10\}, \{2,3\}$ with the $(min, mean, max)$ of p-value equal to $(0, 0.005, 0.012)$. This small difference shows that the performance of QILS is almost insensitive (or not that sensitive) to the size of the action set. The point that QILS is not very sensitive to the size of actions becomes more and more important and useful when there is no sufficient prior knowledge regarding the performance of individual operators. Accordingly, QILS can be executed with a set of different operators and not necessarily the best ones; then, the proposed Q-learning based selection mechanism is able to automatically select the most appropriate operators, i.e. actions, during the search process among all available operators. In addition, by comparing the results of QILS with its corresponding RILS, two important facts are revealed. First, QILS is always better than RILS and can reach better solutions. This behavior has also been statistically verified. Second, the higher the size of action set, the worse the performance of RILS. Indeed, as much as the size of action set increases, the best operators are selected less in RILS. In this situation, a part of the search process in RILS is wasted to perform less performing operators.

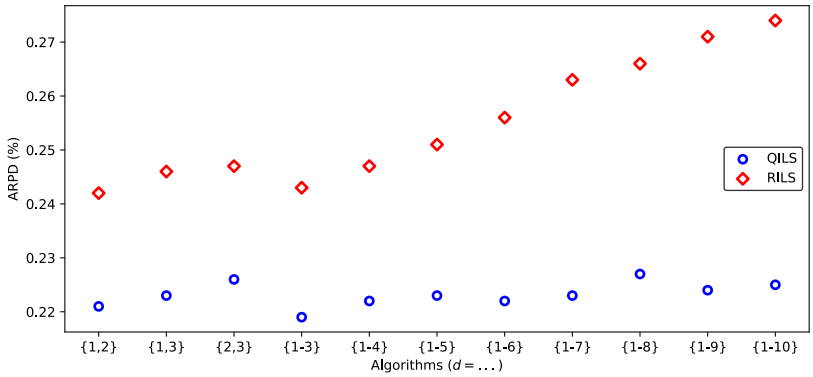

Figure 23 – Performance comparison of QILS and RILS based on ARPD (%) for different sizes of the action set on *Taillard* dataset

Reward function – In this section, the aim is to investigate the effectiveness of the reward mechanism in the proposed QILS compared to the basic 0/1 reward mechanism in the literature. Table 30 compares the performance of QILS with its current reward mechanism based on local/global improvements against the 0/1 reward mechanism on *Taillard* dataset.

Table 30 – Results of comparing QILS with the proposed local/global reward mechanism to QILS with 0/1 reward mechanism for scale $t = 120$ on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.013, 0.044)$.

Inst.		Algorithms			
		QILS with 0/1 reward mechanism		QILS with local/global reward mechanism	
n	m	AvS	BS	AvS	BS
20	5	0.036	0	**0.027**	0
20	10	0.012	0	**0.002**	0
20	20	0.003	0	**0**	0
50	5	0.001	0	**0**	0
50	10	0.413	0.273	**0.372**	0.255
50	20	0.503	0.237	0.494	0.193
100	5	0.008	0.008	0.008	0.008
100	10	0.068	0.019	**0.048**	0.016
100	20	0.736	0.397	**0.703**	0.386
200	10	0.042	0.029	**0.040**	0.024
200	20	0.774	0.492	**0.738**	0.480
500	20	0.330	0.221	**0.307**	0.220
Average		0.245	0.140	**0.228**	0.132

Table 31 – Performance comparison of QILS and RILS for different actions on *Taillard* dataset. The $(min, mean, max)$ of statistical significant p-values with 95% of confidence interval is equal to $(0, 0.004, 0.042)$.

Inst.		Algorithms																
set		$QILS_{1,2}$			$RILS_{1,2}$			$QILS_{1,3}$			$RILS_{1,3}$			$QILS_{2,3}$			$RILS_{2,3}$		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
20	5	0.038	0	3.1	0.041	0.041	1.8	0.038	0	1.9	0.038	0	1.8	0.038	0	2.7	0.041	0.041	2.6
20	10	0.027	0	0.7	0.029	0	2.2	0.033	0	2.3	0.037	0	2.1	0.027	0	3.1	0.032	0	2.9
20	20	0.015	0	3.2	0.017	0	6.4	0.015	0	6.6	0.017	0	6.2	0.010	0	7.7	0.011	0	7.3
50	5	**0.003**	0	7.4	0.001	0	7.3	0.003	0	7.8	0.003	0	7.3	0.003	0	9.9	0.003	0	9.4
50	10	**0.406**	0.269	22.2	0.428	0.282	26.4	**0.407**	0.273	31.0	0.441	0.303	30.7	**0.437**	0.276	27.8	0.468	0.306	26.4
50	20	0.540	0.242	81.1	0.549	0.277	45.6	0.536	0.200	46.2	0.554	0.240	47.1	**0.527**	0.226	63.7	0.562	0.285	61.7
100	5	**0.008**	0.008	15.4	0.009	0.008	22.5	0.009	0.008	27.3	0.010	0.008	25.9	**0.008**	0.008	28.0	0.010	0.008	26.8
100	10	**0.055**	0.016	35.8	0.066	0.019	57.7	**0.056**	0.017	56.8	0.071	0.025	55.6	**0.050**	0.019	65.0	0.066	0.019	64.1
100	20	**0.611**	0.357	193.6	0.669	0.447	187.6	**0.616**	0.362	186.5	0.683	0.407	194.4	**0.654**	0.388	258.2	0.709	0.478	252.7
200	10	**0.048**	0.030	55.1	0.052	0.036	73.9	**0.047**	0.027	77.8	0.053	0.028	75.3	**0.047**	0.031	95.9	0.051	0.035	94.4
200	20	**0.657**	0.426	417.3	0.729	0.487	377.3	**0.660**	0.428	360.5	0.740	0.455	375.5	**0.665**	0.447	438.2	0.734	0.489	444.0
500	20	**0.290**	0.199	949.7	0.351	0.234	761.1	**0.302**	0.209	798.3	0.340	0.224	888.0	**0.289**	0.196	803.2	0.321	0.220	887.1
Average		**0.225**	0.129	148.7	0.245	0.152	130.8	**0.227**	0.127	133.6	0.249	0.141	142.5	**0.229**	0.133	150.3	0.251	0.157	156.6

Inst.		Algorithms																
set		$QILS_{1-3}$			$RILS_{1-3}$			$QILS_{1-4}$			$RILS_{1-4}$			$QILS_{1-5}$			$RILS_{1-5}$		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
20	5	0.036	0	1.8	0.039	0	1.7	0.038	0	2.0	0.038	0	1.9	0.038	0	2.0	0.038	0	1.9
20	10	0.027	0	2.2	0.029	0	2.0	0.024	0	2.4	0.026	0	2.3	0.031	0	2.4	0.031	0	2.3
20	20	0.007	0	5.3	0.010	0	5.0	0.013	0	6.7	0.015	0	6.3	0.015	0	5.8	0.016	0	5.5
50	5	0.002	0	7.3	0.002	0	6.9	0.003	0	8.0	0.002	0	7.6	0.002	0	7.7	0.002	0	7.4
50	10	**0.401**	0.255	24.6	0.428	0.279	24.0	**0.415**	0.286	31.8	0.456	0.323	31.2	**0.414**	0.273	26.0	0.471	0.320	26.2
50	20	0.538	0.239	44.7	0.556	0.304	45.9	**0.539**	0.274	48.0	0.582	0.280	49.8	**0.546**	0.280	47.6	0.591	0.331	51.0
100	5	0.008	0.008	21.7	0.008	0.008	20.6	**0.008**	0.008	27.8	0.010	0.008	26.2	**0.009**	0.008	23.2	0.011	0.008	22.5
100	10	**0.052**	0.016	48.9	0.079	0.021	47.6	**0.053**	0.019	59.5	0.058	0.021	58.5	**0.053**	0.019	50.2	0.063	0.021	50.3
100	20	**0.608**	0.315	185.3	0.671	0.411	192.4	**0.611**	0.391	195.7	0.691	0.441	208.4	**0.607**	0.371	190.1	0.702	0.385	210.2
200	10	**0.043**	0.024	76.9	0.046	0.033	74.9	**0.046**	0.033	80.8	0.051	0.037	79.1	**0.045**	0.028	76.3	0.049	0.029	75.7
200	20	**0.662**	0.455	334.7	0.749	0.493	349.5	**0.665**	0.422	384.2	0.739	0.524	403.9	**0.664**	0.460	336.7	0.750	0.504	358.4
500	20	**0.288**	0.202	812.9	0.338	0.218	913.0	**0.290**	0.208	848.9	0.331	0.237	939.6	**0.294**	0.211	754.7	0.327	0.240	837.4
Average		**0.223**	0.126	130.5	0.246	0.147	140.3	**0.225**	0.137	141.3	0.250	0.156	151.2	**0.226**	0.137	126.9	0.254	0.153	137.4

Inst.		Algorithms																
		$QILS_{1-6}$			$RILS_{1-6}$			$QILS_{1-7}$			$RILS_{1-7}$			$QILS_{1-8}$			$RILS_{1-8}$		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
20	5	0.038	0	2.1	0.039	0	2.0	0.041	0.041	2.6	0.041	0.041	2.5	0.038	0	1.9	0.039	0	1.8
20	10	0.028	0	2.6	0.033	0	2.4	0.031	0	3.1	0.036	0	2.9	0.026	0	2.4	0.035	0	2.3
20	20	0.013	0	6.1	0.013	0	5.8	0.012	0	7.5	0.012	0	7.2	0.012	0	5.9	0.012	0	5.7
50	5	0.002	0	8.2	0.002	0	7.7	0.002	0	10.0	0.002	0	9.5	0.002	0	7.7	0.002	0	7.3
50	10	**0.412**	0.266	32.8	0.488	0.316	33.0	**0.413**	0.236	28.1	0.478	0.320	28.3	**0.428**	0.283	26.2	0.503	0.293	27.1
50	20	**0.538**	0.258	49.9	0.600	0.268	53.8	**0.525**	0.234	59.1	0.624	0.312	64.4	**0.540**	0.274	46.7	0.631	0.323	52.5
100	5	**0.009**	0.008	24.0	0.011	0.008	23.4	**0.008**	0.008	28.2	0.011	0.008	26.8	**0.009**	0.008	23.2	0.012	0.008	22.5

To be continued ...

Table 31 (continued)

Inst.		Algorithms																	
set		$QILS_{1,2}$			$RILS_{1,2}$			$QILS_{1,3}$			$RILS_{1,3}$			$QILS_{2,3}$			$RILS_{2,3}$		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
100	10	**0.052**	0.019	52.8	0.062	0.021	52.7	**0.049**	0.016	64.7	0.060	0.021	64.1	**0.061**	0.016	53.3	0.066	0.019	53.5
100	20	**0.610**	0.377	199.3	0.720	0.427	224.6	**0.615**	0.342	238.5	0.770	0.458	268.4	**0.621**	0.358	192.1	0.746	0.479	223.7
200	10	0.047	0.025	80.2	0.048	0.035	79.7	**0.045**	0.031	101.0	0.050	0.031	99.6	**0.042**	0.029	81.3	0.050	0.036	81.1
200	20	**0.672**	0.469	352.6	0.773	0.515	380.6	**0.690**	0.452	431.3	0.788	0.524	470.4	**0.689**	0.488	353.9	0.799	0.556	391.8
500	20	**0.288**	0.204	867.9	0.326	0.234	963.5	**0.290**	0.208	857.1	0.321	0.227	949.2	**0.293**	0.217	819.3	0.339	0.239	895.4
Average		**0.226**	0.136	139.9	0.260	0.152	152.4	**0.227**	0.131	152.6	0.266	0.162	166.1	**0.230**	0.139	134.5	0.269	0.163	147.1

Inst.		Algorithms											
		$QILS_{1-9}$			$RILS_{1-9}$			$QILS_{1-10}$			$RILS_{1-10}$		
n	m	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)	AvS	BS	T (s)
20	5	0.039	0	2.4	0.041	0.041	2.3	0.038	0	2.1	0.039	0	2.0
20	10	0.031	0	3.0	0.038	0	2.8	0.033	0	2.5	0.033	0	2.4
20	20	0.011	0	6.2	0.014	0	6.0	0.014	0	6.0	0.014	0	5.8
50	5	0.002	0	10.0	0.002	0	9.4	0.002	0	8.2	0.002	0	7.9
50	10	**0.420**	0.256	27.1	0.500	0.303	27.9	**0.426**	0.269	33.2	0.498	0.350	34.2
50	20	**0.532**	0.259	59.2	0.667	0.325	66.3	**0.541**	0.274	49.0	0.672	0.320	56.1
100	5	**0.009**	0.008	30.9	0.011	0.008	29.7	**0.010**	0.008	24.0	0.013	0.008	23.4
100	10	**0.052**	0.019	54.2	0.079	0.021	54.5	**0.058**	0.021	53.8	0.080	0.021	54.5
100	20	**0.616**	0.345	244.1	0.767	0.493	280.2	**0.614**	0.388	197.2	0.786	0.493	234.0
200	10	**0.046**	0.025	102.5	0.051	0.036	101.1	**0.045**	0.034	83.4	0.053	0.034	82.6
200	20	**0.674**	0.460	362.6	0.802	0.472	402.8	**0.662**	0.445	362.3	0.810	0.588	402.9
500	20	**0.296**	0.190	913.9	0.327	0.231	1025.1	**0.297**	0.197	924.9	0.329	0.220	1049.5
Average		**0.227**	0.130	151.3	0.275	0.161	167.3	**0.228**	0.136	145.6	0.277	0.170	162.9

Indeed, in our proposed reward mechanism, each operator receives a reward based on the amount of improvements that occurs in both the best found solution (i.e., global) and the current local optimum (i.e., local) due to its employment. This mechanism allows the operator to be evaluated both locally and globally. However, in the 0/1 reward mechanism, an operator receives a reward equal to 1, if a global improvement is obtained and 0 otherwise, where the amount of the improvement is neglected. As a result, two operators that improve the best found solution receive the same reward equal to 1, regardless of the amount of improvement. The results of Table 30 show that for all instance sets, the QILS framework with the proposed local/global reward mechanism outperforms the QILS framework with 0/1 reward mechanism and achieves better solutions. This shows the effectiveness of considering the amount of local and global improvements into the reward mechanism.

4.5.3 Complexity analysis

The worst-case complexity of the proposed QILS framework is analyzed according to Algorithm 2. In the following, we provide the complexities of the sub-functions used in this algorithm:

- `initialSolution`: The initial solution can be generated randomly, in a greedy manner, or using a heuristic, each of which may have a different complexity.
- `applyOperator`: The complexity of this function depends on the type of the operator used and on the recurring or a single application of the operator to the solution.
- `terminationCriterion`: The choice of a termination criterion is left to the user. In case of using the total execution time, the number of iterations or the number of non-improving iterations, this can be done in $O(1)$;
- `Q-learning`: The only required operation in this function is to find a maximum Q-value from a list of size $|A|$. Accordingly, the complexity of this function is $O(|A|)$;
- `accept`: The complexity of this function depends on the type of the employed acceptance strategy.

Let Itr_{ILS} and Itr_{QILS} ($Itr_{ILS} = Itr_{QILS} \times E$) denote the number of iterations performed by the `while` loops of ILS and QILS, respectively. Considering the complexities of the sub-functions, the (detailed) worst-case complexity of Algorithm 2, $O(QILS)$, is given as Equation 4.10:

$$\begin{aligned} O(QILS) \approx\ & O(\texttt{initialSolution}) + \\ & O(\mathcal{OPT}_l) + \\ & Itr_{QILS}\Big(E\Big(O(\mathcal{OPT}_p) + O(\mathcal{OPT}_l)\Big) + O(\texttt{Q-learning})\Big) \end{aligned} \tag{4.10}$$

Similarly, the (detailed) worst-case complexity of the simple ILS algorithm (without the integration of Q-learning), $O(ILS)$ is given as Equation :

$$\begin{aligned} O(QILS) \approx\ & O(\texttt{initialSolution})+ \\ & O(\mathcal{OPT}_l)+ \\ & Itr_{ILS}\Big(O(\mathcal{OPT}_p) + O(\mathcal{OPT}_l)\Big) \end{aligned} \tag{4.11}$$

Considering that E is a constant, the overhead complexity (i.e., the extra complexity required to use QILS instead of ILS) is equal to $Itr_{QILS} \times |A|$. As a consequence, the complexity overhead of QILS compared to ILS mainly depends on the number of iterations Itr_{QILS} (not on the size of the problem instances) and it imposes only $O(|A|) = O(1)$ extra computations per iteration since $|A|$ is a given constant. In practice, QILS even provides better results with even less computational effort for large instances due to faster convergence toward good (near-optimal/optimal) solutions. The above experiments and results are evidence to support this fact.

The worst-case complexity of the QILS for both applications to TSP and PFSP based on our algorithmic choices (initial solution generation method, local search and perturbation operators, acceptance strategy) is given in Table 32.

Table 32 – Complexity of QILS

App.	QILS to select	$O(\texttt{initialSolution})$	$O(\mathcal{OPT}_l)$	$O(\mathcal{OPT}_p)$	$O(\texttt{Q-learning})$
TSP	Local search	$O(\|V\|^2)$	$O(\|V\|^3)$	$O(1)$	$O(\|A\|)$
	Perturbation	$O(\|V\|^2)$	$O(\|V\|^3)$	$O(1)$	$O(\|A\|)$
PFSP	Perturbation	$O(\|N\|^2\|M\|)$	$O(\|N\|^2\|M\|)$	$O(\|A\| + (\|N\| - \|A\|)^2\|M\| + \|A\|\|N\|\|M\|)$	$O(\|A\|)$

The set of actions in TSP when we use QILS for perturbation operator selection is a tuple $A = (p, R)$, where $p \in \mathbb{P} = \{\mathcal{OPT}_l, \mathcal{OPT}_p\}$ is the type of the perturbation operator and R is the repetition number of the perturbation operator P. Accordingly, the size of action set $|A| = |\mathbb{P}| \times |R|$. Considering that there are a set of multiple local search and perturbation operators, we estimate the $\mathrm{O}(\mathcal{OPT}_l)$ and $\mathrm{O}(\mathcal{OPT}_p)$ functions based on the worst-case complexity of the existing operators.

To conclude the chapter, we remind you that we first proposed a general framework that integrates Q-learning into ILS algorithm and is able to automatically select the search operators without the need for expert knowledge. The proposed framework is applicable to any MHs for solving COPs when a variety of efficient and competing search operators exists. Afterwards, we investigated the performance of the proposed framework through its application to two COPs, TSP and PFSP. In both applications, we observed that the integration of Q-learning into the ILS algorithm yields promising results compared to non-learning version of the ILS algorithm. However, the proposed framework showed the state-of-the-art behavior when solving the PFSP. Despite the generality of the proposed framework to be applied for any MHs, there is still a need to characterize the framework for different COPs, including search operator identifications and parameter tuning. These concerns will be discussed in more detail in the next chapter.

Chapter 5

Conclusions and Future Research

In this thesis, we contributed to an interdisciplinary research domain between Operations Research and Computer Science to investigate how ML techniques can help for automatic design of advanced MHs. This integration is for the aim to guide the MHs toward making better decisions and consequently make MHs more efficient and improve their performance in terms of solution quality, convergence rate, and robustness. The contribution of this thesis to the domain of integrating ML techniques into MHs is an *analytical-technical* contribution.

From an *analytical* viewpoint, we provided, for the first time in the literature, a comprehensive yet technical review on the whole studies addressing this integration. Through this review, we proposed a taxonomy for different ways of integration (i.e., including *algorithm selection*, *fitness evaluation*, *initialization*, *evolution* categorized into *operator selection*, *learnable evolution model*, and *neighbor generation*, *parameter setting*, and *cooperation*) and classified the papers corresponding to each class of integration. Furthermore, we provided a complete analysis and discussion on technical details (i.e., challenges, advantages, disadvantages, perspectives, etc.) of different integrations.

From a *technical* viewpoint, we focused on a particular integration and addressed the problem of *adaptive operator selection* in MHs using ML techniques. To address this problem, we proposed a general framework that integrates the Q-learning algorithm into MHs to adaptively and dynamically select the most appropriate search operators at each step of the search process based on their history of performance. The proposed framework can be applied to any MHs and is mostly recommended when MHs benefit from multiple competitive operators for solving a COP. The reason is that all operators would contribute to the search process and the selection process is not biased toward a single operator (i.e., the most competitive one). In this condition, Q-learning is used to automatically select the most appropriate operator (from a pool of competitive operators) depending on the state of the search and the operator's history of performance.

The employment of the proposed framework is mostly highlighted when there is a rise in the number and variety of problem-specific operators (heuristics) for efficiently solving optimization problems. Selecting and applying these operators within a MH requires much expertise in the domain. That is especially the case for COPs with plenty of proposed problem-specific operators as well as classical operators. In order to efficiently select the operators, one must have knowledge over both the classical and the problem-specific operators. This issue highlights the necessity of an automatic approach to select the most appropriate operator(s) based on their performance without having an expertise in the domain. In this regard, we showed that Q-learning helps users to automatically select operators without any need for the human knowledge. In this way, even inexperienced users are able to select appropriate operators for solving COPs.

We investigated the performance of the proposed framework through its application to two COPs, TSP and PFSP. In both applications, the framework yields significant improvement – in terms of solution quality and convergence rate – compared to its non-learning version, where operators are selected randomly. In other words, the knowledge gathered through the learning process throughout the search process is precious and leads to identifying the most appropriate operator at each decision point and consequently to the success of the framework in converging faster to better solutions. Besides, we also observed that the higher the size of instances, the higher the performance of our algorithm. The main reason can be attributed to a faster convergence of the framework toward better solutions.

Through a sensitivity analysis on the size of the actions, we also observed that the proposed framework is almost insensitive (or very slightly sensitive) to the size of the action

set. This insensitivity becomes more and more important and useful when there is no sufficient prior knowledge regarding the performance of individual operators. Accordingly, the framework can be executed with a set of different operators and not necessarily the best ones; then, the proposed Q-learning based selection mechanism is able to select the operators in a way that the most competitive ones favor.

As another important point, we showed that adding Q-learning does not introduce significant complexity to the framework. Considering the worst-case complexity, the Q-learning introduces a complexity overhead of size of the action set which is a finite constant, and it does not depend on the size of the problem. This complexity overhead becomes even negligible when the size of the action set is small. In practice, we showed that the proposed framework obtains even better results with fewer computational effort, especially for larger instances due to faster convergence toward good (near-optimal/optimal) solutions. This conclusion can be generalized to any MH used in the framework, and also application to any COP.

In terms of the adaptiveness of the proposed framework, we showed how Q-learning is able to adapt the MH's behavior to the characteristics of the search space by selecting the most appropriate operators during the search process. In other words, Q-learning is able to identify the best decision (operator) based on the properties of the problem instance at-hand (the properties of the search space and landscape) merely through a trial and error learning process and without the need to be trained for each specific problem instance a priori. This property can eliminate an extra computational effort for training phase, and assure the user to be applied to any problem instance without any modification.

Besides all the advantages that the proposed framework provides, the integration of RL into MHs carries a set of challenges and limitations. Foremost, this integration introduces a set of additional parameters that need to be tuned/controlled and accordingly more computational effort would be required for parameter setting. For instance, the Q-learning algorithm introduces five more parameters to the framework that need to be carefully tuned. However, we observed that the advantages that the Q-learning brings in terms of obtaining significantly better solutions in even less computational time prevail over this particular challenge.

Another important challenge in Q-learning algorithm is defining a set of states depending on the goal of its usage. The states should be completely descriptive of the problem status to allow selecting the correct action. There are three ways to define the states. The states could be 1) *search-dependent* that reflect the properties of the search process such as the number of non-improving iterations, 2) *problem-dependent* that reflect the properties of the problem through generic features, or 3) *instance-dependent* that reflect the properties of the problem instance such as the number of bins in a bin packing problem [Wau+13]. In this thesis, we have used a search-dependent state. However, other type of states can be defined as a future research direction to evaluate their performance compared to the performance of the proposed framework.

Another important challenge is when the number of available operators for a COP domain increases. For each problem domain, there are numerous search operators available in the literature which show different behavior in solving different instances, and it is difficult to predict how they would behave in solving at-hand instances. There would be three ways to deal with many operators.

The first and simplest approach is to include all available operators in the list of candidate operators among which the selection would be performed. This leads to a high number

of actions in the Q-learning algorithm that results in a higher complexity. When the number of actions increases and due to the use of ϵ-greedy strategy for action selection, there would be an ϵ probability of selection even for inefficient actions. Accordingly, we expect that the overall performance of the algorithm would degrade, as some operators may yield no or little improvement to the solution process. More importantly, when the number of states and actions increases significantly, it would not be possible in practice to infer the Q-value of new states from already explored states due to a large amount of memory needed to save and update the table and also a large amount of time required to explore each state. To overcome this challenge, Q-learning can be replaced with deep Q-learning. The idea behind deep Q-learning is to approximate the Q-values using machine learning techniques such as neural network instead of directly inferring the Q-values from the Q-table.

The second approach is to reduce the number of operators by determining a subset of efficient operators and only including them in the set of candidate operators. This approach requires a pre-processing phase to identify the most efficient operators. There are different ways to identify the efficient operators. One may exhaustively execute MHs with every individual operator, solve the problem instances with these individual operators, and then select the best performing operators. One of the deficiencies of this approach is that it ignores the points that the efficiency of the operators changes according to the region of the search space that is currently being explored due to the non-stationarity of COPs' search space. As a result, one may omit good performing operators base on their premature performance in the pre-processing phase. Also, the computational efforts of the pre-procession phase should not be ignored.

The two above-mentioned approaches possess their particular limitations and challenges. They either add a significant complexity to the framework (first approach) or require an extra and time-consuming pre-processing effort.Therefore, the use of simple yet efficient techniques is recommended. In this regard, the third approach – which we are currently working on – is to consider a dynamic set of candidate operators, instead of a static set with fixed operators, to manage a large set of operators in an online manner. The idea is that the operators found to be inefficient at some stages of the search are momentarily disabled by leaving the candidate set for a fixed number of iterations, and other new operators might be invited to be included in the candidate set for only particular steps of the search. This mechanism is able to include effective and exclude ineffective operator at different stages of the search to attain the highest performance without imposing extra computational overhead. This mechanism is based on the idea of adaptive operator management (AOM) inspired from [Mat+11; MLS10]. Using this approach, despite the second approach, we do not delete any poor operator from the beginning since they may show good performance later on, and at the same time we do not need to spend computational effort to do a pre-processing. To realize this approach, I was invited for a sabbatical leave of three months (May–July 2022) at Polytechnique MontrÃĺal (MontrÃĺal, Canada) to collaborate with Professor Andrea Lodi and Professor Nadia Lahrichi. Until now, we have designed the framework and the next step would be to apply this framework to the PFSP instances to investigate its performance and compare it to our current proposed framework.

Apart from the challenges of the proposed framework, one limitation of this work is that the proposed framework has been only applied to two COPs. These applications and the obtained results could be a good starting point that show the efficiency of Q-learning for operator selection. However, to generalize these conclusions, there is a need to do more research and apply the proposed framework to solve different COPs. Accordingly, we advocate more research on different COPs to validate these results on other problems

as well. As the first step toward this path, I was offered a research visit of two months (November & December 2021) at the University of Nottingham (Nottingham, England) to collaborate with Professor Ender Ozcan, we the aim to investigate the generalizability of the results on other COP domains. During this research visit, first, we assessed whether the proposed framework can be seen as a hyper-heuristic. In fact, operator selection has its root in hyper-heuristics. Hyper-heuristics are high-level automated search methodologies that automate the design and selection of heuristics in solving different problems. Despite meta-heuristics, hyper-heuristics perform search over the space of a set of heuristics, instead of directly searching over the solution space. Given a problem instance and a set of heuristics, hyper-heuristics are able to automatically select/generate appropriate heuristics at each step during the search process. Since our proposed framework is general considering that it can be integrated into any MH, and can be applied to any COP, we aimed to investigate its performance over different COP domains.

The framework was therefore implemented in HyFlex (Hyper-heuristics Flexible framework) a Java interface for design and development of hyper-heuristic. HyFlex was initially developed for the first Cross-Domain Heuristic Search Challenge, CHeSC 2011 whose aim was to encourage researchers to design hyper-heuristic algorithms able to be well generalized across different problem domains, to provide a common ground for the implementation and comparison of different participants' algorithms. From then on, most studies developing hyper-heuristics have implemented their algorithms in HyFlex and therefore, it is a good interface allowing to compare the performance of newly developed hyper-heuristics with the existing ones. HyFlex includes five different problem domains including Boolean Satisfiability (SAT), Bin Packing (BP), Permutation Flowshop Scheduling Problem (PFSP), Traveling Salesman Problem (TSP), and Vehicle Routing Problem (VRP), a set of instances and a set of low-level heuristics for each domain. In HyFlex all the information about a problem domain is hidden to the user and the low-level heuristics are black-boxes whose type is the only information available to the user.

We have executed our framework on a set of instances for CHeSC 2011 challenge (5 instances per problem domain). We also followed the same rules as CheSC 2011. Accordingly, our algorithm has been run 31 independent times for each instance and the median result for each instance has been recorded. We compared our algorithm with 20 developed hyper-heuristics of CheSC 2011 and also three state-of-the-art hyper-heuristics from the literature [CWL18; KÖ16; DÖB15]. The set of algorithms within the comparison are then given a score based on their median values for each instance using a formula 1 point scoring system. Based on this system, for each instance, the top performing algorithm receives 10 points, the subsequent receives 8 points, and then 6,4,2, and 1 points. If a tie occurs, the corresponding point is equally shared between each algorithm. The algorithms are then ranked based on their total score over thirty instances across all six problem domains.

We have performed initial experiments to obtain initial results. Using the CHeSC 2011 scoring system, the score of our algorithm for each problem domain as well as the total score over all problem domains has been recorded. Accordingly, the proposed framework (i.e., QILS) has initially obtained the highest scores for SAT and PFSP, as we expected for PFSP, and in total, the QILS framework has obtained the rank 4 with a total score of 83.0. This shows that our approach can perform well also as a hyper-heuristic, and it is competitive to the state-of-the-art hyper-heuristics. Despite meta-heuristics that need to be tailored to a specific domain and once tailored to a specific domain they may not be efficient to solve problems from other domains, hyper-heuristics are able to deal

with different variety of problem domains with only being tailored once.

At the end of this chapter, we are interested to go beyond the integration of ML techniques to MHs for the purpose of operator selection and discuss some general challenges and limitations about the integration of ML techniques into MHs regardless of its purposes. Furthermore, promising research directions are also proposed, where ML techniques can help to deal with optimization problems.

In this integration, it is obvious that the higher the volume of data, the higher the performance of ML techniques. Data availability is indeed an important challenge when integrating ML techniques into MHs. In fact, collecting or even generating enough data is a hard task. Even if enough historical data is available, the way of sampling from historical data to appropriately mimic the behavior reflected in such data is another challenge [BLP21a]. One way of tackling the data availability challenge could be using *Few-Shot Learning* to train a model with a very small amount of training data [Wan+20a]. By using prior knowledge of similar problem instances, few-shot learning can be rapidly generalized to new tasks containing only a few problem instances with supervised information.

The majority of studies in the literature use conventional ML techniques such as k-NN, k-means, SVMs, LR, etc. With the rapid development of new technologies, real-world problems are becoming increasingly complex, and with the new advances in digitalization, various real-time data are collected massively that cannot be processed by classical ML techniques. Such big data carries several issues that need to be taken into consideration [Emr16]. To cope with such big data, more advanced ML techniques such as deep learning can be integrated into MHs. In this regard, when various ML techniques are available to be integrated into MHs for a particular purpose, the algorithm selection problem can be studied to select the most appropriate ML technique. Also, with the development of supercomputers, it could be an interesting future research direction to explore the parallelism concept in the integration of ML techniques into MHs using GPU (Graphics Processing Units) and TPUs (Tensor Processing Units) accelerators [CMT04; Alb05; VLMT11].

Two important issues to consider in real-world optimization problems are the uncertainty of input data, particularly when the input data statistically contains various distributions, and the dynamicity of the input data. For data uncertainty, one promising research direction could be using ML techniques such as clustering methods (e.g., k-means, SOM) to cluster the input data with the aim of discriminating the data with different distributions. These classes of data can be then used/integrated to solve the optimization problem at hand. For the dynamicity of data, ML techniques can be used to monitor/predict the evolution of the input data, and once a new evolution is detected by ML techniques, the optimization variables are updated correspondingly.

Although there are lots of studies in the literature studying each way of integration, the trade-off between the gains vs. the extra computational effort of using ML techniques has not been explicitly studied in most studied. Also, some classes such as *Initialization* (see Section 3.4) has been studied only for a few COP domains. Accordingly, another future research direction is to investigate using the integration of ML techniques into MHs for other COP domains as well as complex optimization problems such as multi-objective optimization, bi-level optimization, etc., to investigate the applicability and potential gain of this integration, and also to check whether the obtained conclusions can be validated on/generalized to other COPs as well. This direction also opens other research questions that are worthy for further investigations.

Almost all the studies in the literature only deal with the integration of ML techniques into MHs with a single purpose, while the higher performance of MHs is expected to be achieved when ML techniques serve MHs for multiple purposes. Therefore, an interesting future research direction could be integrating ML techniques into MHs simultaneously for different purposes. For instance, implementing parameter control and adaptive operator selection simultaneously in a MH may increase the overall performance of the MH throughout the search process.

Publications

The scientific achievements of this research work have been presented to the community through several journal articles and oral presentations in national and international conference.

[Journal articles]:

- **Karimi-Mamaghan, M.**, Mohammadi, M., Meyer, P., & Pasdeloup, B. (2022). Learning to select operators in meta-heuristics for solving combinatorial optimization problems: An application to flowshop scheduling problem. *European Journal of Operational Research*. In press.
- **Karimi-Mamaghan, M.**, Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M., & Talbi, E. G. (2022). Machine Learning at the service of Meta-heuristics for solving Combinatorial Optimization Problems: A state-of-the-art. *European Journal of Operational Research*, 296(2), 393-422.
- **Karimi-Mamaghan, M.**, Pasdeloup, B., Mohammadi, M., Meyer, P. (2021). A Learning-Based Iterated Local Search Algorithm for Solving the Traveling Salesman Problem. In: Dorronsoro, B., Amodeo, L., Pavone, M., Ruiz, P. (eds) Optimization and Learning. OLA 2021. *Communications in Computer and Information Science*, vol 1443. Springer, Cham.
- **Karimi-Mamaghan, M.**, Mohammadi, M., Jula, P., Pirayesh, A., & Ahmadi, H. (2020). A learning-based metaheuristic for a multi-objective agile inspection planning model under uncertainty. *European Journal of Operational Research*, 285(2), 513-537.
- **Karimi-Mamaghan, M.**, Mohammadi, M., Pirayesh, A., Karimi-Mamaghan, A. M., & Irani, H. (2020). Hub-and-spoke network design under congestion: A learning based metaheuristic. *Transportation Research Part E: Logistics and Transportation Review*, 142, 102069.

[Conference presentations]:

- **Karimi-Mamaghan, M.**, Pasdeloup, B., Meyer, P., Mohammadi, M. Learning to select operators in meta-heuristics: Application to flowshop scheduling problem. In Canadian Operational Research Society Annual Conference (CORS 2022), June 2022, Vancouver, Canada.
- **Karimi-Mamaghan, M.**, Meyer, P., Mohammadi, M., Pasdeloup, B. A reinforcement learning-based operator selection in iterated local search for solving scheduling problems. In 31st European Conference on Operational Research (EURO 2021), July 2021, Athens, Greece.
- **Karimi-Mamaghan, M.**, Pasdeloup, B., Mohammadi, M., & Meyer, P. A Learning-based Iterated Local Search Algorithm for Solving the Traveling Salesman Problem. In OLA 2021: 4th International Conference on Optimization and learning, June 2021, Italy (online).

- **Karimi-Mamaghan, M.**, Pasdeloup, B., Meyer, P., Mohammadi, M. Learning to select the local search and perturbation operators of meta-heuristics for solving the Traveling Salesman Problem. In Canadian Operational Research Society Annual Conference (CORS 2021), June 2021, Canada (Online).
- **Karimi-Mamaghan, M.**, Mohammadi, M., Pasdeloup, B., Billot, R., & Meyer, P. An Online Learning-based Metaheuristic for Solving Combinatorial Optimization Problems. In ROADEF 2020: 21th Annual Conference of the French Society of Operations Research and Decision Aiding, February 2020, Montpelier, France.

Appendices

A.1 List of COPs

Table A.1 – Exhaustive list and the abbreviation (Abv.) of the COPs studied by the articles reviewed in this thesis

COP	Abv.	Description
Assignment Problem	AP	Assigning a set of locations to a set of facilities such that the total assignment cost is minimized [DGVDC18].
Arc Routing Problem	ARP	Finding a set of tours with minimum cost in an undirected graph to serve the positive demand of edges by a set of available vehicles where each tour begins and ends at the depot [CY14].
Assemble-To-Order Problem	ATOP	Determining an order based on which the parts and sub-assemblies are made but the final assembly is delayed until the customer orders are received such that the total production cost is minimized [Hor+13].
Berth Allocation Problem	BAP	Allocating the berthing position and berthing time to the incoming vessels to perform loading/unloading activities such that the total vessels' waiting time or the early or delayed departures is minimized [WDS20].
Bin-Packing Problem	BPP	Placing N items in a number of capacitated knapsacks so that the total number of knapsacks used to pack all items is minimized [Bur+11].
Capacitated Lot Sizing Problem	CLSP	Planning the lot size of a set of different items over a planning horizon under production capacity constraints such that the total production, setup, and inventory cost is minimized [CR09].
Facility Location Problem	FLP	Locating a number of facilities in a set of potential locations to serve a set of customers with predefined locations such that the total opening and transportation cost is minimized [MKC10].
Flowshop Scheduling Problem	FSP	Finding the order of processing N jobs on M machines with the same sequence such that the makespan, total tardiness, or total lag between the jobs is minimized [PKD18].
Graph Coloring Problem	GCP	Finding the minimum number of colors for coloring the vertices of a graph such that no two adjacent vertices have the same color or finding the maximum subgraph of a graph to be colored with k colors such that no two adjacent vertices have the same color [ZHD16; MKN20].
Heat Exchanger Design Problem	HEDP	Designing the structure of tubes under technical and environmental constraints including the size of the exchanger and air temperature such that the heat transfer rate is maximized [Dom+04].
Hub Location Problem	HLP	Locating a set of hubs and allocating a set of origin and destination nodes to the located hubs for transferring the origin-destination flows through the hubs such that the total hub opening and transportation costs is minimized [MJTM19].
Inspection Planning Problem	IPP	Determining which quality characteristics of a product should be inspected at which stage of the production process such that the total inspection cost is minimized [KM+20a].
Job-Shop Scheduling	JSP	scheduling the processing of N jobs consists of a sequence of tasks that need to be performed in a given order on specific subsets of M machines such that the makespan or total tardiness is minimized [Nas+19].
Knapsack problem	KP	Placing a number of items with specific values and dimension in M capacitated knapsacks such that the total value of the knapsacks is maximized [CGM09].

To be continued ...

Table A.1 (continued)

COP	Abv.	Description
Location-Routing Problem	LRP	Locating a number of facilities in a set of potential locations, assigning customers with predefined demand to the located facilities, and finding the routes from located facilities to customers such that the total cost of opening facilities, cost of vehicles and transportation cost is minimized [Zha+16a].
Maximum Satisfiability Problem	MAX-SAT	Finding an assignment of the truth values to the variables of a Boolean formula such that the number of satisfied clauses is maximized [Mir+18].
Mixed-model Assembly Line Sequencing Problem	MASP	Determining the optimal production planning of multiple products along a single assembly line while maintaining the least possible inventories [MGS20].
Multiprocessor Scheduling Problem	MSP	Given a directed graph representing a parallel program, where the vertices represent the tasks and the edges represent the communication cost and task dependencies, scheduling the tasks on a network of processors under task precedence constraints such that the makespan is minimized [LA16].
Nurse Rostering Problem	NRP	Assigning nurses to working shifts under a set of constraints including nurse preferences, time restrictions, labor legislation, and hospital standards such that the total cost of the hospital is minimized or the nurses' preferences are maximized [GB17].
Orienteering Problem	OP	Selecting a set of nodes from available nodes with specific score and determining the shortest path between the selected nodes such that the total score of the visited nodes is maximized [GLL18].
Pickup and Delivery Problem	PDP	Designing a set of routes to collect commodities from specific origins and deliver them to their specific destinations using capacitated vehicles such that the total cost is minimized [LT12].
Parallel Machine Scheduling Problem	PMSP	Scheduling the processing of N jobs on M identical parallel machines such that the total makespan is minimized [Sil+19].
Project Scheduling Problem	PSP	Assigning limited resources (employees) to activities with predefined duration and resource requirements under activity precedence relations such that the lateness or the total tardiness is minimized [PSS08].
Personnel Scheduling Problem	PSSP	Assigning personnel to working shifts under a set of constraints (e.g., shift time) such that the total cost is minimized [Bur+11].
Quadratic Assignment Problem	QAP	A special case of Assignment Problem with quadratic objective function [PBA13].
Boolean Satisfiability Problem	SAT	Determining if the variables of a Boolean formula can be substituted by *True* and *False* values such that the Boolean formula turns out to be *TRUE* (Satisfiable) or not [MS08].
Single-Machine Scheduling Problem	SMSP	A special case of Flow-Shop scheduling problem where there is only a single machine [TZ19].
Set Packing Problem	SPP	Determining/picking a subset of elements from a bigger set such that the total value of picking is maximized [RPM06].
Job Sequencing and Tool Switching Problem	SSP	Sequencing N jobs, each of which requires a predefined set of tools, on a single flexible machine and assigning tools to a capacitated machine such that the number of tool switches is minimized [Ahm+18].
Transmission Expansion Planning Problem	TEPP	Planning new transmission facilities as an expansion to the existing transmission network to satisfy demand without load interruption under technical constraints such that the total investment and operational cost is minimized [SFH16].
Traveling Salesman Problem	TSP	Finding a tour in a complete weighted graph that goes through all vertices only once and returns to the starting vertex such that the tour cost is minimized [Kan+16].
Timetabling Problem	TTP	Allocating predefined resources (teachers and rooms) to events (classes) such that there is no conflict between any two events and a set of objectives are satisfied [RR+21].
Vehicle Routing Problem	VRP	Finding a set of undirected edges in a graph by which the demand of customers located in the vertices is satisfied by a set of vehicles that visit each customer exactly once. Vehicles start and end their route at the depot such that the total transportation cost is minimized [GR+19].
Vertex Separator Problem	VSP	Partitioning the graph into three non-empty subsets A, B, and C such that there is no edge between A and B, and $\|C\|$ is minimized subject to a bound on $max\{\|A\|, \|B\|\}$ [BEB17].
Winner Determination Problem	WDP	Considering a set of bids in a combinatorial auction, assigning items to bidders such that the auctioneerâĂŹs revenue is maximized [SJG18].

To be continued ...

Table A.1 (continued)

COP	Abv.	Description
Water Distribution System Design Problem	WDSDP	Determining the location, size, and capacity of water system components including pipes and pumps such that the system's reliability (ability to supply adequate water with acceptable pressure and quality to customers) is maximized [LCA11].
Weighted Independent Domination Problem	WIDP	Determining a pairwise non-adjacent subset D of V of a graph $G = (V, E)$ such that every vertex not in D is adjacent to at least one vertex in D [Wan+20b].
Workforce Scheduling and Routing Problem	WSRP	Assigning workforce to the activities needed to be performed at different locations where the workforce need to travel between locations to perform the activities such that the employees travel time or hiring cost is minimized [LSRVMG18].

A.2 Best-known solutions for PFSP

Table A.2 – Best-known solutions (C^*_{max}) vs. the best C_{max} found by QILS for each instance over 30 runs for time scale $t = 120$ of *Taillard* dataset. LB and UB stand for lower bound and upper bound, respectively.

Instance (tai#)			C^*_{max}		Best	Instance (tai#)			C^*_{max}			Instance (tai#)			C^*_{max}		Best
#	n	m	LB	UB	C_{max}	#	n	m	LB	UB	C_{max}	#	n	m	LB	UB	C_{max}
001	20	5	1278	1278	1278	041	50	10	2991	2991	3023	081	100	20	6106	6202	6234
002	20	5	1359	1359	1359	042	50	10	2867	2867	2870	082	100	20	6183	6183	6206
003	20	5	1081	1081	1081	043	50	10	2839	2839	2852	083	100	20	6252	6271	6284
004	20	5	1293	1293	1293	044	50	10	3063	3063	3063	084	100	20	6254	6269	6303
005	20	5	1235	1235	1235	045	50	10	2976	2976	2977	085	100	20	6262	6314	6343
006	20	5	1195	1195	1195	046	50	10	3006	3006	3006	086	100	20	6302	6364	6379
007	20	5	1234	1234	1234	047	50	10	3093	3093	3101	087	100	20	6184	6268	6283
008	20	5	1206	1206	1206	048	50	10	3037	3037	3038	088	100	20	6315	6401	6423
009	20	5	1230	1230	1230	049	50	10	2897	2897	2902	089	100	20	6204	6275	6305
010	20	5	1108	1108	1108	050	50	10	3065	3065	3078	090	100	20	6404	6434	6464
011	20	10	1582	1582	1582	051	50	20	3771	3850	3863	091	200	10	10862	10862	10870
012	20	10	1659	1659	1659	052	50	20	3668	3704	3708	092	200	10	10480	10480	10485
013	20	10	1496	1496	1496	053	50	20	3591	3640	3642	093	200	10	10922	10922	10922
014	20	10	1377	1377	1377	054	50	20	3635	3720	3730	094	200	10	10889	10889	10889
015	20	10	1419	1419	1419	055	50	20	3553	3610	3611	095	200	10	10524	10524	10526
016	20	10	1397	1397	1397	056	50	20	3667	3681	3689	096	200	10	10326	10326	10330
017	20	10	1484	1484	1484	057	50	20	3672	3704	3711	097	200	10	10854	10854	10857
018	20	10	1538	1538	1538	058	50	20	3627	3691	3700	098	200	10	10730	10730	10733
019	20	10	1593	1593	1593	059	50	20	3645	3743	3750	099	200	10	10438	10438	10438
020	20	10	1591	1591	1591	060	50	20	3696	3756	3767	100	200	10	10675	10675	10676
021	20	20	2297	2297	2297	061	100	5	5493	5493	5493	101	200	20	11152	11195	11224
022	20	20	2099	2099	2099	062	100	5	5268	5268	5268	102	200	20	11143	11203	11279
023	20	20	2326	2326	2326	063	100	5	5171	5171	5175	103	200	20	11281	11281	11382
024	20	20	2223	2223	2223	064	100	5	5014	5014	5014	104	200	20	11275	11275	11322
025	20	20	2291	2291	2291	065	100	5	5250	5250	5250	105	200	20	11259	11259	11288
026	20	20	2226	2226	2226	066	100	5	5135	5135	5135	106	200	20	11176	11176	11224
027	20	20	2273	2273	2273	067	100	5	5246	5246	5246	107	200	20	111337	11360	11425
028	20	20	2200	2200	2200	068	100	5	5094	5094	5094	108	200	20	11301	11334	11387
029	20	20	2237	2237	2237	069	100	5	5448	5448	5448	109	200	20	11145	11192	11241
030	20	20	2178	2178	2178	070	100	5	5322	5322	5322	110	200	20	11284	11288	11332
031	50	5	2724	2724	2724	071	100	10	5770	5770	5770	111	500	20	26040	26059	26112
032	50	5	2834	2834	2834	072	100	10	5349	5349	5349	112	500	20	26500	26520	26634
033	50	5	2621	2621	2621	073	100	10	5676	5676	5676	113	500	20	26371	26371	26435
034	50	5	2751	2751	2751	074	100	10	5781	5781	5781	114	500	20	26456	26456	26520
035	50	5	2863	2863	2863	075	100	10	5467	5467	5467	115	500	20	26334	26334	26366
036	50	5	2829	2829	2829	076	100	10	5303	5303	5303	116	500	20	26469	26477	26522
037	50	5	2725	2725	2725	077	100	10	5595	5595	5596	117	500	20	26389	26389	26412
038	50	5	2683	2683	2683	078	100	10	5617	5617	5621	118	500	20	26560	26560	26619
039	50	5	2552	2552	2552	079	100	10	5871	5871	5875	119	500	20	26005	26005	26075
040	50	5	2782	2782	2782	080	100	10	5845	5845	5845	120	500	20	26457	26457	26513

Table A.3 – Best-known solutions (C^*_{max}) vs. the best C_{max} found by QILS for each instance over 30 runs for time scale $t = 120$ of *VRF-hard-small* dataset. LB and UB stand for lower bound and upper bound, respectively.

Instance (vrf#)			C^*_{max}		Best	Instance (vrf#)			C^*_{max}		Best	Instance (vrf#)			C^*_{max}		Best
#	n	m	LB	UB	C_{max}	#	n	m	LB	UB	C_{max}	#	n	m	LB	UB	C_{max}
001	10	5	523	695	695	081	30	5	1727	1805	1805	161	50	5	2970	3055	3055
002	10	5	556	698	698	082	30	5	1510	1575	1575	162	50	5	2784	2853	2853
003	10	5	588	728	728	083	30	5	1608	1673	1673	163	50	5	2682	2746	2746
004	10	5	565	697	697	084	30	5	1716	1781	1781	164	50	5	2773	2836	2836
005	10	5	578	713	713	085	30	5	1645	1707	1707	165	50	5	2806	2866	2866
006	10	5	617	748	748	086	30	5	1807	1875	1875	166	50	5	2782	2841	2841
007	10	5	602	728	728	087	30	5	1686	1749	1749	167	50	5	2548	2600	2600
008	10	5	568	683	683	088	30	5	1646	1706	1706	168	50	5	2631	2684	2684
009	10	5	633	761	761	089	30	5	1674	1735	1735	169	50	5	2570	2621	2621
010	10	5	554	664	664	090	30	5	1582	1637	1637	170	50	5	2781	2834	2834
011	10	10	797	1097	1097	091	30	10	1721	1944	1943	171	50	10	2746	2926	2930
012	10	10	845	1146	1146	092	30	10	1860	2098	2098	172	50	10	2841	3035	3035
013	10	10	831	1124	1124	093	30	10	1857	2077	2077	173	50	10	2836	3019	3016
014	10	10	769	1038	1038	094	30	10	1747	1945	1945	174	50	10	2838	3003	3009
015	10	10	817	1093	1093	095	30	10	1818	2023	2023	175	50	10	3070	3252	3252
016	10	10	812	1085	1085	096	30	10	1830	2043	2043	176	50	10	2973	3149	3149
017	10	10	839	1115	1115	097	30	10	1767	1967	1967	177	50	10	2722	2842	2865
018	10	10	840	1113	1113	098	30	10	1701	1896	1896	178	50	10	2932	3072	3075
019	10	10	789	1045	1045	099	30	10	1712	1908	1908	179	50	10	2858	3022	3022
020	10	10	832	1099	1099	100	30	10	1722	1915	1915	180	50	10	2906	3056	3060
021	10	15	921	1307	1307	101	30	15	1999	2381	2378	181	50	15	2988	3316	3322
022	10	15	988	1399	1399	102	30	15	1952	2318	2317	182	50	15	3037	3347	3359
023	10	15	996	1398	1398	103	30	15	1950	2304	2304	183	50	15	2998	3301	3301
024	10	15	1041	1452	1452	104	30	15	2079	2444	2444	184	50	15	3192	3521	3520
025	10	15	992	1373	1373	105	30	15	2062	2423	2421	185	50	15	3039	3334	3337
026	10	15	964	1329	1329	106	30	15	1968	2306	2306	186	50	15	3042	3346	3355
027	10	15	1049	1445	1445	107	30	15	1978	2316	2316	187	50	15	3181	3490	3493
028	10	15	1048	1443	1443	108	30	15	2019	2366	2366	188	50	15	3135	3430	3430
029	10	15	1058	1428	1428	109	30	15	1929	2259	2259	189	50	15	2928	3205	3205
030	10	15	1085	1461	1461	110	30	15	2047	2385	2385	190	50	15	3104	3399	3406
031	10	20	1191	1652	1652	111	30	20	2119	2643	2643	191	50	20	3164	3693	3686
032	10	20	1273	1759	1759	112	30	20	2284	2835	2835	192	50	20	3224	3719	3713
033	10	20	1254	1726	1726	113	30	20	2265	2783	2783	193	50	20	3284	3784	3778
034	10	20	1236	1678	1678	114	30	20	2213	2680	2680	194	50	20	3231	3709	3707
035	10	20	1259	1700	1700	115	30	20	2205	2672	2672	195	50	20	3157	3632	3629
036	10	20	1400	1889	1889	116	30	20	2245	2715	2715	196	50	20	3295	3795	3787
037	10	20	1251	1678	1678	117	30	20	2244	2712	2712	197	50	20	3219	3696	3696
038	10	20	1235	1655	1655	118	30	20	2328	2812	2812	198	50	20	3295	3783	3783
039	10	20	1280	1706	1706	119	30	20	2318	2795	2795	199	50	20	3337	3816	3809
040	10	20	1248	1663	1663	120	30	20	2329	2805	2805	200	50	20	3301	3769	3767
041	20	5	1095	1192	1192	121	40	5	2292	2396	2396	201	60	5	3276	3350	3350
042	20	5	1173	1275	1275	122	40	5	2351	2442	2442	202	60	5	2990	3054	3054
043	20	5	1224	1323	1323	123	40	5	2106	2174	2174	203	60	5	3147	3214	3214
044	20	5	1047	1127	1127	124	40	5	2082	2149	2149	204	60	5	3202	3266	3266
045	20	5	1244	1339	1339	125	40	5	2179	2247	2247	205	60	5	3139	3197	3197
046	20	5	994	1066	1066	126	40	5	2091	2154	2154	206	60	5	3058	3107	3107
047	20	5	1078	1154	1154	127	40	5	2143	2207	2207	207	60	5	3263	3315	3315
048	20	5	1030	1102	1102	128	40	5	2350	2414	2414	208	60	5	3386	3438	3438
049	20	5	1231	1317	1317	129	40	5	2247	2305	2305	209	60	5	3074	3121	3121
050	20	5	1164	1243	1243	130	40	5	2289	2348	2348	210	60	5	3608	3663	3663
051	20	10	1290	1532	1532	131	40	10	2258	2480	2480	211	60	10	3256	3435	3441
052	20	10	1292	1525	1525	132	40	10	2237	2444	2444	212	60	10	3489	3655	3664
053	20	10	1352	1592	1592	133	40	10	2244	2412	2412	213	60	10	3261	3423	3419
054	20	10	1226	1442	1442	134	40	10	2313	2472	2471	214	60	10	3305	3455	3461
055	20	10	1371	1604	1604	135	40	10	2276	2425	2425	215	60	10	3359	3505	3506
056	20	10	1348	1576	1576	136	40	10	2387	2547	2547	216	60	10	3448	3594	3594
057	20	10	1363	1591	1591	137	40	10	2344	2501	2501	217	60	10	3501	3654	3654
058	20	10	1353	1574	1574	138	40	10	2338	2491	2490	218	60	10	3402	3552	3552
059	20	10	1312	1530	1530	139	40	10	2275	2411	2411	219	60	10	3529	3685	3685
060	20	10	1279	1489	1489	140	40	10	2337	2478	2481	220	60	10	3346	3492	3492
061	20	15	1525	1936	1936	141	40	15	2624	3011	3005	221	60	15	3623	3940	3947
062	20	15	1526	1905	1905	142	40	15	2491	2821	2821	222	60	15	3582	3888	3888
063	20	15	1453	1798	1798	143	40	15	2572	2906	2904	223	60	15	3590	3880	3883
064	20	15	1466	1813	1813	144	40	15	2576	2919	2915	224	60	15	3413	3716	3718
065	20	15	1517	1875	1875	145	40	15	2615	2945	2946	225	60	15	3611	3881	3880
066	20	15	1587	1960	1960	146	40	15	2494	2805	2811	226	60	15	3598	3893	3888
067	20	15	1566	1933	1933	147	40	15	2555	2868	2863	227	60	15	3529	3809	3813
068	20	15	1477	1822	1822	148	40	15	2578	2900	2897	228	60	15	3456	3749	3746
069	20	15	1575	1940	1940	149	40	15	2407	2708	2705	229	60	15	3512	3800	3807
070	20	15	1511	1861	1861	150	40	15	2625	2945	2948	230	60	15	3610	3902	3902
071	20	20	1741	2270	2270	151	40	20	2770	3326	3322	231	60	20	3689	4163	4158
072	20	20	1682	2170	2170	152	40	20	2697	3226	3224	232	60	20	3804	4290	4281
073	20	20	1772	2277	2277	153	40	20	2713	3233	3228	233	60	20	3870	4365	4357
074	20	20	1685	2165	2165	154	40	20	2724	3233	3233	234	60	20	3731	4193	4187
075	20	20	1736	2225	2225	155	40	20	2571	3055	3052	235	60	20	3747	4196	4194
076	20	20	1790	2291	2291	156	40	20	2699	3192	3187	236	60	20	3764	4202	4200
077	20	20	1785	2282	2282	157	40	20	2749	3244	3244	237	60	20	3818	4263	4256
078	20	20	1707	2178	2178	158	40	20	2764	3266	3261	238	60	20	3758	4180	4186
079	20	20	1851	2354	2354	159	40	20	2834	3335	3332	239	60	20	3764	4221	4217
080	20	20	1732	2199	2199	160	40	20	2653	3122	3117	240	60	20	3779	4202	4187

Table A.4 – Best-known solutions (C^*_{max}) vs. the best C_{max} found by QILS for each instance over 30 runs for time scale $t = 120$ of *VRF-hard-large* dataset. LB and UB stand for lower bound and upper bound, respectively.

Instance (vrf#)			C^*_{max}		Best	Instance (vrf#)			C^*_{max}		Best	Instance (vrf#)			C^*_{max}		Best
#	n	m	LB	UB	C_{max}	#	n	m	LB	UB	C_{max}	#	n	m	LB	UB	C_{max}
001	100	20	5705	6198	6200	081	300	60	17876	20522	20508	161	600	40	32219	33839	33714
002	100	20	5836	6306	6299	082	300	60	17750	20399	20315	162	600	40	31978	33467	33433
003	100	20	5771	6238	6234	083	300	60	17894	20434	20350	163	600	40	32316	33866	33711
004	100	20	5783	6245	6250	084	300	60	17837	20395	20340	164	600	40	32162	33693	33572
005	100	20	5876	6296	6286	085	300	60	17791	20341	20239	165	600	40	32071	33553	33467
006	100	20	5913	6321	6307	086	300	60	17814	20388	20351	166	600	40	32290	33809	33677
007	100	20	6004	6434	6428	087	300	60	17935	20457	20440	167	600	40	32203	33686	33641
008	100	20	5694	6104	6088	088	300	60	17918	20410	20411	168	600	40	32039	33482	33353
009	100	20	5928	6354	6345	089	300	60	18000	20549	20475	169	600	40	32228	33697	33536
010	100	20	5766	6145	6148	090	300	60	17937	20472	20444	170	600	40	32235	33642	33504
011	100	40	6611	7881	7853	091	400	20	20727	21120	21068	171	600	60	33093	36198	36066
012	100	40	6738	8007	7980	092	400	20	21092	21457	21432	172	600	60	33108	36184	35940
013	100	40	6698	7935	7901	093	400	20	21133	21441	21432	173	600	60	33159	36201	36103
014	100	40	6715	7932	7920	094	400	20	20942	21247	21251	174	600	60	33225	36136	36085
015	100	40	6778	8011	7997	095	400	20	21203	21553	21552	175	600	60	33093	36153	36089
016	100	40	6816	8023	8011	096	400	20	20944	21214	21195	176	600	60	33093	36116	36061
017	100	40	6793	8006	8006	097	400	20	21331	21625	21605	177	600	60	33215	36179	36068
018	100	40	6807	7979	7956	098	400	20	21029	21277	21276	178	600	60	33206	36185	36061
019	100	40	6758	7931	7901	099	400	20	21059	21346	21313	179	600	60	33273	36195	36139
020	100	40	6774	7952	7932	100	400	20	21235	21538	21533	180	600	60	33177	36163	36078
021	100	60	7502	9395	9361	101	400	40	22002	23578	23494	181	700	20	36078	36394	36385
022	100	60	7720	9596	9567	102	400	40	21903	23456	23375	182	700	20	36032	36337	36327
023	100	60	7523	9349	9331	103	400	40	22099	23575	23498	183	700	20	36287	36568	36502
024	100	60	7596	9426	9417	104	400	40	21897	23409	23377	184	700	20	36188	36452	36413
025	100	60	7638	9465	9437	105	400	40	21843	23339	23302	185	700	20	36380	36584	36594
026	100	60	7809	9667	9637	106	400	40	22011	23444	23396	186	700	20	36376	36671	36627
027	100	60	7576	9391	9352	107	400	40	22078	23556	23488	187	700	20	36381	36624	36618
028	100	60	7697	9534	9529	108	400	40	21921	23411	23378	188	700	20	36287	36522	36504
029	100	60	7706	9527	9500	109	400	40	22109	23637	23579	189	700	20	36091	36329	36312
030	100	60	7774	9598	9568	110	400	40	22257	23720	23692	190	700	20	36282	36417	36398
031	200	20	10928	11305	11317	111	400	60	22867	25607	25518	191	700	40	37286	38964	38796
032	200	20	10898	11265	11250	112	400	60	22869	25656	25654	192	700	40	37211	38775	38624
033	200	20	10958	11327	11323	113	400	60	23079	25821	25743	193	700	40	37080	38621	38559
034	200	20	10857	11208	11186	114	400	60	23097	25837	25756	194	700	40	37232	38785	38737
035	200	20	10861	11208	11190	115	400	60	23077	25877	25771	195	700	40	37125	38671	38479
036	200	20	10981	11367	11355	116	400	60	22802	25536	25503	196	700	40	37186	38710	38464
037	200	20	11034	11380	11382	117	400	60	22868	25600	25493	197	700	40	36959	38585	38395
038	200	20	10783	11141	11105	118	400	60	23058	25800	25657	198	700	40	37508	39059	38872
039	200	20	10773	11123	11111	119	400	60	23108	25882	25807	199	700	40	37261	38814	38604
040	200	20	10965	11310	11310	120	400	60	23104	25767	25683	200	700	40	37419	38850	38748
041	200	40	11812	13132	13162	121	500	20	26071	26411	26403	201	700	60	38230	41436	41311
042	200	40	11735	13102	13082	122	500	20	26358	26681	26653	202	700	60	38173	41375	41101
043	200	40	11879	13264	13253	123	500	20	26116	26409	26378	203	700	60	38127	41317	41224
044	200	40	11862	13232	13169	124	500	20	25844	26124	26091	204	700	60	38287	41401	41288
045	200	40	11662	13043	12955	125	500	20	26505	26781	26748	205	700	60	38212	41262	41203
046	200	40	11788	13124	13120	126	500	20	26152	26443	26443	206	700	60	38305	41340	41202
047	200	40	11926	13299	13166	127	500	20	26163	26433	26413	207	700	60	37882	40876	40785
048	200	40	11864	13238	13163	128	500	20	26062	26318	26305	208	700	60	38427	41474	41387
049	200	40	11817	13166	13068	129	500	20	26199	26442	26446	209	700	60	38212	41291	41179
050	200	40	11921	13228	13195	130	500	20	25838	26072	26065	210	700	60	38368	41377	41269
051	200	60	12824	14990	14950	131	500	40	26940	28548	28476	211	800	20	41256	41558	41533
052	200	60	12862	14954	14930	132	500	40	27162	28793	28687	212	800	20	41145	41407	41371
053	200	60	12996	15200	15134	133	500	40	27064	28607	28516	213	800	20	41159	41425	41380
054	200	60	12925	15044	14967	134	500	40	27231	28828	28691	214	800	20	41157	41426	41426
055	200	60	13012	15130	15058	135	500	40	27109	28683	28604	215	800	20	41457	41710	41710
056	200	60	12952	15035	14997	136	500	40	26903	28524	28437	216	800	20	41795	42010	41970
057	200	60	12958	15040	15016	137	500	40	27162	28760	28662	217	800	20	41138	41425	41418
058	200	60	12947	14968	14962	138	500	40	27215	28698	28597	218	800	20	41241	41492	41462
059	200	60	12878	15022	14946	139	500	40	27298	28870	28750	219	800	20	41517	41796	41796
060	200	60	12892	15000	14959	140	500	40	27225	28758	28730	220	800	20	41331	41574	41568
061	300	20	15773	16149	16123	141	500	60	27846	30861	30785	221	800	40	42144	43671	43559
062	300	20	16123	16512	16510	142	500	60	27819	30828	30811	222	800	40	42197	43746	43634
063	300	20	15835	16173	16160	143	500	60	28045	31125	30958	223	800	40	42180	43749	43596
064	300	20	15860	16181	16152	144	500	60	28054	30928	30987	224	800	40	42360	43892	43705
065	300	20	15987	16342	16356	145	500	60	27936	30935	30872	225	800	40	42473	43905	43746
066	300	20	15804	16137	16103	146	500	60	28091	31027	31006	226	800	40	42258	43811	43600
067	300	20	15932	16266	16259	147	500	60	27962	30928	30929	227	800	40	42242	43766	43554
068	300	20	16093	16416	16402	148	500	60	28089	30988	30984	228	800	40	42362	43839	43660
069	300	20	16059	16376	16339	149	500	60	28098	30978	30915	229	800	40	42341	43879	43619
070	300	20	16517	16899	16850	150	500	60	28190	31050	30987	230	800	40	42335	43861	43674
071	300	40	16764	18298	18259	151	600	20	31124	31433	31389	231	800	60	43230	46470	46363
072	300	40	17015	18454	18465	152	600	20	31124	31418	31394	232	800	60	43189	46493	46441
073	300	40	16983	18457	18391	153	600	20	31186	31429	31436	233	800	60	43271	46389	46317
074	300	40	16914	18351	18305	154	600	20	31254	31547	31509	234	800	60	43274	46457	46339
075	300	40	17070	18484	18431	155	600	20	31204	31448	31414	235	800	60	43319	46401	46276
076	300	40	16978	18449	18431	156	600	20	31470	31717	31714	236	800	60	43210	46421	46302
077	300	40	17049	18419	18433	157	600	20	31254	31527	31527	237	800	60	43207	46319	46206
078	300	40	16927	18392	18341	158	600	20	31299	31564	31461	238	800	60	43358	46474	46332
079	300	40	16987	18394	18345	159	600	20	31320	31577	31528	239	800	60	43406	46538	46470
080	300	40	17007	18401	18332	160	600	20	30885	31130	31108	240	800	60	43236	46244	46295

Bibliography

[AA16] Deepanshu Arora and Gopal Agarwal. "Meta-heuristic approaches for flowshop scheduling problems: a review". In: *International Journal of Advanced Operations Management* 8.1 (2016), pp. 1–16.

[AB+18] Mohamed Abdel-Basset, Gunasekaran Manogaran, Heba Rashad, and Abdel Nasser H Zaied. "A comprehensive review of quadratic assignment problem: variants, hybrids and applications". In: *Journal of Ambient Intelligence and Humanized Computing* (2018), pp. 1–24.

[ACBF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3 (2002), pp. 235–256.

[AD+18] Fatemah Al-Duoli, Ghaith Rabadi, Mamadou Seck, and Holly A Handley. "Hybridizing Meta-RaPS with Machine Learning Algorithms". In: *2018 IEEE Technology and Engineering Management Conference (TEMSCON)*. IEEE. 2018, pp. 1–6.

[AEK19] Ismail M Ali, Daryl Essam, and Kathryn Kasmarik. "New Designs of k-means Clustering and Crossover Operator for Solving Traveling Salesman Problems using Evolutionary Algorithms". In: *Proceedings of the 11th International Joint Conference on Computational Intelligence*. 2019, pp. 123–130.

[AEK20] Ismail M Ali, Daryl Essam, and Kathryn Kasmarik. "A novel design of differential evolution for solving discrete traveling salesman problems". In: *Swarm and Evolutionary Computation* 52 (2020), p. 100607.

[AGL07] Esmaeil Atashpaz-Gargari and Caro Lucas. "Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition". In: *2007 IEEE congress on evolutionary computation*. Ieee. 2007, pp. 4661–4667.

[Ahm+18] Ehsan Ahmadi, Boris Goldengorin, Gürsel A Süer, and Hadi Mosadegh. "A hybrid method of 2-TSP and novel learning-based GA for job sequencing and tool switching problem". In: *Applied Soft Computing* 65 (2018), pp. 214–229.

[Alb05] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. Vol. 47. John Wiley & Sons, 2005.

[Ale12] Aldeida Aleti. "An adaptive approach to controlling parameters of evolutionary algorithms". In: *Swinburne University of Technology* (2012).

[Ale+14] Aldeida Aleti, Irene Moser, Indika Meedeniya, and Lars Grunske. "Choosing the appropriate forecasting model for predictive parameter control". In: *Evolutionary computation* 22.2 (2014), pp. 319–349.

[Ali+18] Mir Mohammad Alipour, Seyed Naser Razavi, Mohammad Reza Feizi Derakhshi, and Mohammad Ali Balafar. “A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem”. In: *Neural Computing and Applications* 30.9 (2018), pp. 2935–2951.

[All15] Ali Allahverdi. “The third comprehensive survey on scheduling problems with setup times/costs”. In: *European Journal of Operational Research* 246.2 (2015), pp. 345–378.

[AM13] Aldeida Aleti and Irene Moser. “Entropy-based adaptive range parameter control for evolutionary algorithms”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013, pp. 1501–1508.

[AMM12] Aldeida Aleti, Irene Moser, and Sanaz Mostaghim. “Adaptive range parameter control”. In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–8.

[AP14] Leanderson André and Rafael Stubs Parpinelli. “A binary differential evolution with adaptive parameters applied to the multiple knapsack problem”. In: *Mexican International Conference on Artificial Intelligence*. Springer. 2014, pp. 61–71.

[Arm+06] Warren Armstrong, Peter Christen, Eric McCreath, and Alistair P Rendell. “Dynamic algorithm selection using reinforcement learning”. In: *2006 International Workshop on Integrating AI and Data Mining*. IEEE. 2006, pp. 18–25.

[Arn+21] Florian Arnold, Ítalo Santana, Kenneth Sörensen, and Thibaut Vidal. “PILS: exploring high-order neighborhoods by pattern mining and injection”. In: *Pattern Recognition* (2021), p. 107957.

[AS19] Florian Arnold and Kenneth Sörensen. “What makes a VRP solution good? the generation of problem-specific knowledge for heuristics”. In: *Computers & Operations Research* 106 (2019), pp. 280–288.

[AS+94] Rakesh Agrawal, Ramakrishnan Srikant, et al. “Fast algorithms for mining association rules”. In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. Citeseer. 1994, pp. 487–499.

[AZ02] Eric Angel and Vassilis Zissimopoulos. “On the hardness of the quadratic assignment problem with metaheuristics”. In: *Journal of Heuristics* 8.4 (2002), pp. 399–414.

[Bar10a] Dariusz Barbucha. “Cooperative solution to the vehicle routing problem”. In: *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. Springer. 2010, pp. 180–189.

[Bar10b] Dariusz Barbucha. “Synchronous vs. asynchronous cooperative approach to solving the vehicle routing problem”. In: *International Conference on Computational Collective Intelligence*. Springer. 2010, pp. 403–412.

[Bar+13] Hugo Barbalho, Isabel Rosseti, Simone L Martins, and Alexandre Plastino. “A hybrid data mining GRASP with path-relinking”. In: *Computers & Operations Research* 40.12 (2013), pp. 3159–3173.

[BAW17] Andreas Beham, Michael Affenzeller, and Stefan Wagner. “Instance-based algorithm selection on quadratic assignment problem landscapes”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2017, pp. 1471–1478.

[BB60] George EP Box and Donald W Behnken. “Some new three level designs for the study of quantitative variables”. In: *Technometrics* 2.4 (1960), pp. 455–475.

[BBZ17] Thomas Bartz-Beielstein and Martin Zaefferer. “Model-based methods for continuous and discrete global optimization”. In: *Applied Soft Computing* 55 (2017), pp. 154–167.

[BEB17] Una Benlic, Michael G Epitropakis, and Edmund K Burke. “A hybrid breakout local search and reinforcement learning approach to the vertex separator problem”. In: *European Journal of Operational Research* 261.3 (2017), pp. 803–818.

[Ben+20] Yoshua Bengio, Emma Frejinger, Andrea Lodi, Rahul Patel, and Sriram Sankaranarayanan. “A learning-based algorithm to quickly compute good primal solutions for Stochastic Integer Programs”. In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2020, pp. 99–111.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[BKB14] Arina Buzdalova, Vladislav Kononov, and Maxim Buzdalov. “Selecting evolutionary operators using reinforcement learning: Initial explorations”. In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 1033–1036.

[BLP21a] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. “Machine Learning for Combinatorial Optimization: a Methodological Tour d’Horizon”. In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[BLP21b] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. “Machine learning for combinatorial optimization: A methodological tour dâĂŹhorizon”. In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[BLW86] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[Bos+18] Jakob Bossek, Christian Grimme, Stephan Meisel, Günter Rudolph, and Heike Trautmann. “Local search effects in bi-objective orienteering”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 585–592.

[BPQ06] Edmund K Burke, Sanja Petrovic, and Rong Qu. “Case-based heuristic selection for timetabling problems”. In: *Journal of Scheduling* 9.2 (2006), pp. 115–132.

[BR03] Christian Blum and Andrea Roli. “Metaheuristics in combinatorial optimization: Overview and conceptual comparison”. In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.

[BT16] Jakob Bossek and Heike Trautmann. “Evolving instances for maximizing performance differences of state-of-the-art inexact TSP solvers”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2016, pp. 48–59.

[Bur+11] Edmund K Burke, Michel Gendreau, Gabriela Ochoa, and James D Walker. “Adaptive iterated local search for cross-domain optimisation”. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 2011, pp. 1987–1994.

[Bur+13] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. "Hyper-heuristics: A survey of the state of the art". In: *Journal of the Operational Research Society* 64.12 (2013), pp. 1695–1724.

[Bur+19] Edmund K Burke, Matthew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. "A classification of hyper-heuristic approaches: revisited". In: *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477.

[BW51] George EP Box and Kenneth B Wilson. "On the experimental attainment of optimum conditions". In: *Journal of the royal statistical society: Series b (Methodological)* 13.1 (1951), pp. 1–38.

[Cal+17] Laura Calvet, Jésica de Armas, David Masip, and Angel A Juan. "Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs". In: *Open Mathematics* 15.1 (2017), pp. 261–280.

[CDJ12] David Corne, Clarisse Dhaenens, and Laetitia Jourdan. "Synergies between operations research and data mining: The emerging use of multi-objective approaches". In: *European Journal of Operational Research* 221.3 (2012), pp. 469–479.

[CDM14] David Catteeuw, Madalina Drugan, and Bernard Manderick. "âĂŸGuidedâĂŹRestarts Hill-Climbing". In: *In Search of Synergies between Reinforcement learning and Evolutionary Computation, Workshop at the 13th International Conference on Parallel Problem Solving from Nature. Ed. by Madalina M. Drugan and Bernard Manderick. Ljubljana, Slovenia.* 2014, pp. 1–4.

[CGM09] José Manuel Cadenas, M Carmen Garrido, and E Muñoz. "Using machine learning in a cooperative hybrid parallel strategy of metaheuristics". In: *Information Sciences* 179.19 (2009), pp. 3255–3267.

[Cha17] Yen-Ching Chang. "Using k-means clustering to improve the efficiency of ant colony optimization for the traveling salesman problem". In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2017, pp. 379–384.

[Che+16] Yujie Chen, Peter Ivan Cowling, Fiona AC Polack, and Philip James Mourdjis. "A multi-arm bandit neighbourhood search for routing and scheduling problems". In: *The University of York* (2016), pp. 2–32.

[Che+20] Ronghua Chen, Bo Yang, Shi Li, and Shilong Wang. "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem". In: *Computers & Industrial Engineering* 149 (2020), p. 106778.

[Che+21] Chen-Yang Cheng, Pourya Pourhejazy, Kuo-Ching Ying, and Chen-Fang Lin. "Unsupervised Learning-based Artificial Bee Colony for minimizing non-value-adding operations". In: *Applied Soft Computing* (2021), p. 107280.

[Chr76] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[Chu+19] Xianghua Chu, Fulin Cai, Can Cui, Mengqi Hu, Li Li, and Quande Qin. "Adaptive recommendation model using meta-learning for population-based algorithms". In: *Information Sciences* 476 (2019), pp. 192–210.

[CMT04] Sébastien Cahon, Nordine Melab, and E-G Talbi. "Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics". In: *Journal of heuristics* 10.3 (2004), pp. 357–380.

[CR09] Marco Caserta and E Quiñonez Rico. "A cross entropy-Lagrangean hybrid algorithm for the multi-item capacitated lot-sizing problem with setup times". In: *Computers & Operations Research* 36.2 (2009), pp. 530–548.

[Cro58] Georges A Croes. "A method for solving traveling-salesman problems". In: *Operations research* 6.6 (1958), pp. 791–812.

[CTA05] C Cotta, EG Talbi, and Enrique Alba. "Parallel Hybrid Metaheuristics". In: *Parallel Metaheuristics: A New Class of Algorithms* 47 (2005), p. 347.

[Cun08] Pádraig Cunningham. "Dimension reduction". In: *Machine learning techniques for multimedia*. Springer, 2008, pp. 91–112.

[CWL18] Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. "Automatic design of hyper-heuristic based on reinforcement learning". In: *Information Sciences* 436 (2018), pp. 89–107.

[CWL19] Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. "An artificial bee colony algorithm with a modified choice function for the Traveling Salesman Problem". In: *Swarm and evolutionary computation* 44 (2019), pp. 622–635.

[CY14] Pietro Consoli and Xin Yao. "Diversity-driven selection of multiple crossover operators for the capacitated arc routing problem". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2014, pp. 97–108.

[DB05] Marco Dorigo and Christian Blum. "Ant colony optimization theory: A survey". In: *Theoretical computer science* 344.2-3 (2005), pp. 243–278.

[Deg+16] Hans Degroote, Bernd Bischl, Lars Kotthoff, and Patrick De Causmaecker. "Reinforcement learning for automatic online algorithm selection-an empirical study". In: *ITAT 2016 Proceedings* 1649 (2016), pp. 93–101.

[Der+11] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms". In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 3–18.

[DGVDC18] Hans Degroote, José Luis González-Velarde, and Patrick De Causmaecker. "Applying Algorithm Selection–a Case Study for the Generalised Assignment Problem". In: *Electronic Notes in Discrete Mathematics* 69 (2018), pp. 205–212.

[DLDMN08] Francisco Chagas De Lima, Jorge Dantas De Melo, and Adriao Duarte Doria Neto. "Using the Q-learning algorithm in the constructive phase of the GRASP and reactive GRASP metaheuristics". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 4169–4176.

[DLPS17] Jérémie Dubois-Lacoste, Federico Pagnozzi, and Thomas Stützle. "An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem". In: *Computers & Operations Research* 81 (2017), pp. 160–166.

[DLZ15] Yong Deng, Yang Liu, and Deyun Zhou. "An improved genetic algorithm with initial population strategy for symmetric TSP". In: *Mathematical Problems in Engineering* 2015 (2015), pp. 1–6.

[DMTC17] Alan Díaz-Manríquez, Gregorio Toscano, and Carlos A Coello Coello. "Comparison of metamodeling techniques in evolutionary algorithms". In: *Soft Computing* 21.19 (2017), pp. 5647–5663.

[Dob10] Felix Dobslaw. "A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks". In: *International Conference on Computer Mathematics and Natural Computing*. WASET. 2010, pp. 1–4.

[DÖB15] John H Drake, Ender Özcan, and Edmund K Burke. "A modified choice function hyper-heuristic controlling unary and binary operators". In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2015, pp. 3389–3396.

[Dom+04] Piotr A Domanski, David Yashar, Kenneth A Kaufman, and Ryszard S Michalski. "An optimized design of finned-tube evaporators using the learnable evolution model". In: *Hvac&R Research* 10.2 (2004), pp. 201–211.

[DP18] Augusto Lopez Dantas and Aurora Trinidad Ramirez Pozo. "A meta-learning algorithm selection approach for the quadratic assignment problem". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2018, pp. 1–8.

[DP20] Augusto Dantas and Aurora Pozo. "On the use of fitness landscape features in meta-learning based algorithm selection for the quadratic assignment problem". In: *Theoretical Computer Science* 805 (2020), pp. 62–75.

[DPRVZD10] Ocotlán Díaz-Parra, Jorge A Ruiz-Vanoye, and José C Zavala-Díaz. "Population pre-selection operators used for generating a non-random initial population to solve vehicle routing problem with time windows". In: *Scientific Research and Essays* 5.22 (2010), pp. 3529–3537.

[Dra+19] John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. "Recent advances in selection hyper-heuristics". In: *European Journal of Operational Research* 285.2 (2019), pp. 405–428.

[DT+15] Giacomo Di Tollo, Frédéric Lardeux, Jorge Maturana, and Frédéric Saubion. "An experimental study of adaptive control for evolutionary algorithms". In: *Applied Soft Computing* 35 (2015), pp. 359–372.

[Due93] Gunter Dueck. "New optimization heuristics: The great deluge algorithm and the record-to-record travel". In: *Journal of Computational physics* 104.1 (1993), pp. 86–92.

[EHM99] Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. "Parameter control in evolutionary algorithms". In: *IEEE Transactions on evolutionary computation* 3.2 (1999), pp. 124–141.

[EKEB+18] Mehdi El Krari, Bouazza El Benani, et al. "Breakout Local Search for the Travelling Salesman Problem". In: *Computing and Informatics* 37.3 (2018), pp. 656–672.

[Emr16] Ali Emrouznejad. *Big data optimization: recent developments and challenges*. Vol. 18. Springer, 2016.

[Fai+19] Suthida Fairee, Charoenchai Khompatraporn, Santitham Prom-on, and Booncharoen Sirinaovakul. "Combinatorial Artificial Bee Colony Optimization with Reinforcement Learning Updating for Travelling Salesman Problem". In: *2019 16th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. IEEE. 2019, pp. 93–96.

[Fia+08] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. "Extreme value based adaptive operator selection". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 175–184.

[Fia10] Álvaro Fialho. "Adaptive operator selection for optimization". PhD thesis. UniversitÃľ Paris Sud - Paris XI, 2010.

[FK13] Fedor V Fomin and Petteri Kaski. "Exact exponential algorithms". In: *Communications of the ACM* 56.3 (2013), pp. 80–88.

[FR95] Thomas A Feo and Mauricio GC Resende. "Greedy randomized adaptive search procedures". In: *Journal of global optimization* 6.2 (1995), pp. 109–133.

[Fra+11] Gianpiero Francesca, Paola Pellegrini, Thomas Stützle, and Mauro Birattari. "Off-line and on-line tuning: a study on operator selection for a memetic algorithm applied to the QAP". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2011, pp. 203–214.

[FVF14] Victor Fernandez-Viagas and Jose M Framinan. "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem". In: *Computers & Operations Research* 45 (2014), pp. 60–67.

[FVF19] Victor Fernandez-Viagas and Jose M Framinan. "A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective". In: *Computers & Operations Research* 112 (2019), p. 104767.

[FVMPF20] Victor Fernandez-Viagas, Jose M Molina-Pariente, and Jose M Framinan. "Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling". In: *European Journal of Operational Research* 282.3 (2020), pp. 858–872.

[FVRF17] Victor Fernandez-Viagas, Rubén Ruiz, and Jose M Framinan. "A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation". In: *European Journal of Operational Research* 257.3 (2017), pp. 707–721.

[Gao+16] Shangce Gao, Yirui Wang, Jiujun Cheng, Yasuhiro Inazumi, and Zheng Tang. "Ant colony optimization with clustering for solving the dynamic location routing problem". In: *Applied Mathematics and Computation* 285 (2016), pp. 149–173.

[GB17] Angeliki Gretsista and Edmund K Burke. "An iterated local search framework with adaptive operator selection for nurse rostering". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2017, pp. 93–108.

[GGNS20] Claudio Gambella, Bissan Ghaddar, and Joe Naoum-Sawaya. "Optimization problems for machine learning: A survey". In: *European Journal of Operational Research* (2020). DOI: 10.1016/j.ejor.2020.08.045.

[GJAP19] Daniel González-Juarez and Esther Andrés-Pérez. "Study of the Influence of the Initial a Priori Training Dataset Size in the Efficiency and Convergence of Surrogate-Based Evolutionary Optimization". In: *Evolutionary and Deterministic Methods for Design Optimization and Control With Applications to Industrial and Societal Problems*. Springer, 2019, pp. 181–194.

[GK06] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*. Vol. 57. Springer Science & Business Media, 2006.

[GKL01] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. "A new heuristic optimization algorithm: harmony search". In: *simulation* 76.2 (2001), pp. 60–68.

[GL98] Fred Glover and Manuel Laguna. "Tabu search". In: *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[GLL18] Aldy Gunawan, Hoong Chuin Lau, and Kun Lu. "ADOPT: combining parameter tuning and adaptive operator ordering for solving a class of orienteering problems". In: *Computers & Industrial Engineering* 121 (2018), pp. 82–96.

[Glo86] Fred Glover. "Future paths for integer programming and linds to artificial intelligence". In: *Computers and Operations Research* 13 (1986), pp. 533–549.

[GP10] Michel Gendreau and Jean-Yves Potvin, eds. *Handbook of Metaheuristics*. Springer US, 2010.

[GR+19] Andres E Gutierrez-Rodríguez, Santiago E Conant-Pablos, José C Ortiz-Bayliss, and Hugo Terashima-Marín. "Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning". In: *Expert Systems with Applications* 118 (2019), pp. 470–481.

[GS10] Matteo Gagliolo and Jürgen Schmidhuber. "Algorithm selection as a bandit problem with unbounded losses". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2010, pp. 82–96.

[GY19] T Gocken and M Yaktubay. "COMPARISON OF DIFFERENT CLUSTERING ALGORITHMS VIA GENETIC ALGORITHM FOR VRPTW". In: *International Journal of Simulation Modeling* 18.4 (2019), pp. 574–585.

[Han+14] Stephanus Daniel Handoko, Duc Thien Nguyen, Zhi Yuan, and Hoong Chuin Lau. "Reinforcement learning for adaptive operator selection in memetic search applied to quadratic assignment problem". In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 193–194.

[Hao+16] Jing-hua Hao, Min Liu, Jian-hua Lin, and Cheng Wu. "A hybrid differential evolution approach based on surrogate modelling for scheduling bottleneck stages". In: *Computers & Operations Research* 66 (2016), pp. 215–224.

[Has+18] Ahmad B Hassanat, VB Prasath, Mohammed Ali Abbadi, Salam Amer Abu-Qdari, and Hossam Faris. "An improved genetic algorithm with a new initialization mechanism based on regression techniques". In: *Information* 9.7 (2018), p. 167.

[Hel09] Keld Helsgaun. "General k-opt submoves for the Lin–Kernighan TSP heuristic". In: *Mathematical Programming Computation* 1.2 (2009), pp. 119–163.

[Hem06] Jano I van Hemert. "Evolving combinatorial problem instances that are difficult to solve". In: *Evolutionary Computation* 14.4 (2006), pp. 433–462.

[HL14] Jing-hua Hao and Min Liu. "A surrogate modelling approach combined with differential evolution for solving bottleneck stage scheduling problems". In: *2014 World Automation Congress (WAC)*. IEEE. 2014, pp. 120–124.

[HLY19] Changwu Huang, Yuanxiang Li, and Xin Yao. "A survey of automatic parameter tuning methods for metaheuristics". In: *IEEE Transactions on Evolutionary Computation* 24.2 (2019), pp. 201–216.

[Hol+92] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[Hon+02] Tzung-Pei Hong, Hong-Shung Wang, Wen-Yang Lin, and Wen-Yuan Lee. "Evolution of appropriate crossover and mutation operators in a genetic process". In: *Applied Intelligence* 16.1 (2002), pp. 7–17.

[Hoo11] Holger H Hoos. "Automated algorithm configuration and parameter tuning". In: *Autonomous search*. Springer, 2011, pp. 37–71.

[Hoo+18] Holger H Hoos, Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. "Portfolio-based algorithm selection for circuit QBFs". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2018, pp. 195–209.

[Hor+13] Shih-Cheng Horng, Shin-Yeu Lin, Loo Hay Lee, and Chun-Hung Chen. "Memetic algorithm for real-time combinatorial stochastic simulation optimization problems with performance analysis". In: *IEEE transactions on cybernetics* 43.5 (2013), pp. 1495–1509.

[Hut+06] Frank Hutter, Youssef Hamadi, Holger H Hoos, and Kevin Leyton-Brown. "Performance prediction and automated tuning of randomized and parametric algorithms". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2006, pp. 213–228.

[HW90] Alain Hertz and Dominique de Werra. "The tabu search metaheuristic: how we used it". In: *Annals of Mathematics and Artificial Intelligence* 1.1-4 (1990), pp. 111–121.

[HZG10] M Mehdi S Haghighi, M Hadi Zahedi, and Mehdi Ghazizadeh. "A multi level priority clustering GA based approach for solving heterogeneous Vehicle Routing Problem (PCGVRP)". In: *Innovations and Advances in Computer Sciences and Engineering*. Springer, 2010, pp. 331–335.

[JDT06] Laetitia Jourdan, Clarisse Dhaenens, and El-Ghazali Talbi. "Using datamining techniques to help metaheuristics: A short survey". In: *International Workshop on Hybrid Metaheuristics*. Springer. 2006, pp. 57–69.

[JG10] Fariborz Jolai and A Ghanbari. “Integrating data transformation techniques with Hopfield neural networks for solving travelling salesman problem”. In: *Expert Systems with Applications* 37.7 (2010), pp. 5331–5335.

[Jin05] Yaochu Jin. “A comprehensive survey of fitness approximation in evolutionary computation”. In: *Soft computing* 9.1 (2005), pp. 3–12.

[Jin11] Yaochu Jin. “Surrogate-assisted evolutionary computation: Recent advances and future challenges”. In: *Swarm and Evolutionary Computation* 1.2 (2011), pp. 61–70.

[JKK18] Donghwi Jung, Doosun Kang, and Joong Hoon Kim. “Development of a hybrid harmony search for water distribution system design”. In: *KSCE Journal of Civil Engineering* 22.4 (2018), pp. 1506–1514.

[JM97] David S Johnson and Lyle A McGeoch. “The traveling salesman problem: A case study in local optimization”. In: *Local search in combinatorial optimization* 1.1 (1997), pp. 215–310.

[Joh54] Selmer Martin Johnson. “Optimal two-and three-stage production schedules with setup times included”. In: *Naval research logistics quarterly* 1.1 (1954), pp. 61–68.

[Jos+22] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. “Learning the travelling salesperson problem requires rethinking generalization”. In: *Constraints* (2022), pp. 1–29.

[Jou+05] Laetitia Jourdan, David Corne, Dragan Savic, and Godfrey Walters. “Preliminary investigation of the âĂŸlearnable evolution modelâĂŹfor faster/better multiobjective water systems design”. In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2005, pp. 841–855.

[JPY88] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. “How easy is local search?” In: *Journal of computer and system sciences* 37.1 (1988), pp. 79–100.

[JRR95] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. “The traveling salesman problem”. In: *Handbooks in operations research and management science* 7 (1995), pp. 225–330.

[JS04] Yaochu Jin and Bernhard Sendhoff. “Reducing fitness evaluations using clustering techniques and neural network ensembles”. In: *Genetic and Evolutionary Computation Conference*. Springer. 2004, pp. 688–699.

[Kad+11] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. “Algorithm selection and scheduling”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2011, pp. 454–469.

[Kan+11a] Jorge Kanda, Andre Carvalho, Eduardo Hruschka, and Carlos Soares. “Selection of algorithms to solve traveling salesman problems using meta-learning 1”. In: *International Journal of Hybrid Intelligent Systems* 8.3 (2011), pp. 117–128.

[Kan+11b] Jorge Y Kanda, Andre CPLF de Carvalho, Eduardo R Hruschka, and Carlos Soares. “Using meta-learning to recommend meta-heuristics for the traveling salesman problem”. In: *2011 10th International Conference on Machine Learning and Applications and Workshops*. Vol. 1. IEEE. 2011, pp. 346–351.

[Kan+12] Jorge Kanda, Carlos Soares, Eduardo Hruschka, and Andre De Carvalho. "A meta-learning approach to select meta-heuristics for the traveling salesman problem using MLP-Based label ranking". In: *International Conference on Neural Information Processing*. Springer. 2012, pp. 488–495.

[Kan+16] Jorge Kanda, Andre de Carvalho, Eduardo Hruschka, Carlos Soares, and Pavel Brazdil. "Meta-learning to select the best meta-heuristic for the Traveling Salesman Problem: A comparison of meta-features". In: *Neurocomputing* 205 (2016), pp. 393–406.

[Kar] "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art". In: *European Journal of Operational Research* 296.2 (2022), pp. 393–422. ISSN: 0377-2217.

[Kar05] Dervis Karaboga. *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Citeseer, 2005.

[Ken06] James Kennedy. "Swarm intelligence". In: *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.

[Ker+19] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. "Automated algorithm selection: Survey and perspectives". In: *Evolutionary computation* 27.1 (2019), pp. 3–45.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[Kha+17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. "Learning combinatorial optimization algorithms over graphs". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6348–6358.

[KHE14] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. "Parameter control in evolutionary algorithms: Trends and challenges". In: *IEEE Transactions on Evolutionary Computation* 19.2 (2014), pp. 167–187.

[KIG14] Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Springer Berlin Heidelberg, 2014. DOI: 10.1007/978-3-642-37846-1.

[Kiz+19] Damla Kizilay, Mehmet Fatih Tasgetiren, Quan-Ke Pan, and Liang Gao. "A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion". In: *Algorithms* 12.5 (2019), p. 100.

[KK06] Sotiris Kotsiantis and Dimitris Kanellopoulos. "Association rules mining: A recent overview". In: *GESTS International Transactions on Computer Science and Engineering* 32.1 (2006), pp. 71–82.

[KLM96] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[KM00] Kenneth A Kaufman and Ryszard S Michalski. "Applying learnable evolution model to heat exchanger design". In: *AAAI/IAAI*. 2000, pp. 1014–1019.

[KM+20a] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Payman Jula, Amir Pirayesh, and Hadi Ahmadi. "A learning-based metaheuristic for a multi-objective agile inspection planning model under uncertainty". In: *European Journal of Operational Research* 285.2 (2020), pp. 513–537.

[KM+20b] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Amir Pirayesh, Amir Mohammad Karimi-Mamaghan, and Hassan Irani. "Hub-and-spoke network design under congestion: A learning based metaheuristic". In: *Transportation Research Part E: Logistics and Transportation Review* 142 (2020), p. 102069.

[KM+21] Maryam Karimi-Mamaghan, Bastien Pasdeloup, Mehrdad Mohammadi, and Patrick Meyer. "A Learning-Based Iterated Local Search Algorithm for Solving the Traveling Salesman Problem". In: *Optimization and Learning*. Ed. by Bernabé Dorronsoro, Lionel Amodeo, Mario Pavone, and Patricia Ruiz. Cham: Springer International Publishing, 2021, pp. 45–61. ISBN: 978-3-030-85672-4.

[KM+22] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Bastien Pasdeloup, and Patrick Meyer. "Learning to select operators in meta-heuristics: An integration of Q-learning into the iterated greedy algorithm for the permutation flowshop scheduling problem". In: *European Journal of Operational Research* (2022).

[Kno06] Joshua Knowles. "ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems". In: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), pp. 50–66.

[KÖ16] Ahmed Kheiri and Ender Özcan. "An iterated multi-stage selection hyperheuristic". In: *European Journal of Operational Research* 250.1 (2016), pp. 77–90.

[Koç+16] Çağrı Koç, Tolga Bektaş, Ola Jabali, and Gilbert Laporte. "Thirty years of heterogeneous vehicle routing". In: *European Journal of Operational Research* 249.1 (2016), pp. 1–21.

[Kor+12] Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*. Vol. 2. Springer, 2012.

[Kot14] Lars Kotthoff. "Algorithm selection for combinatorial search problems: A survey". In: *AI Magazine* 35.3 (2014), pp. 48–60.

[Kot16] Lars Kotthoff. "Algorithm selection for combinatorial search problems: A survey". In: *Data Mining and Constraint Programming*. Springer, 2016, pp. 149–190.

[LA16] Nasser Lotfi and Adnan Acan. "A tournament-based competitive-cooperative multiagent architecture for real parameter optimization". In: *Soft Computing* 20.11 (2016), pp. 4597–4617.

[Law01] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[LBCK05] Alexandre Le Bouthillier, Teodor G Crainic, and Peter Kropf. "A guided cooperative search for the vehicle routing problem with time windows". In: *IEEE Intelligent Systems* 20.4 (2005), pp. 36–42.

[LCA11] Stefan Lessmann, Marco Caserta, and Idel Montalvo Arango. "Tuning metaheuristics: A data mining based approach for particle swarm optimization". In: *Expert Systems with Applications* 38.10 (2011), pp. 12826–12838.

[Leó+17a] Alan Dávila de León, Eduardo Lalla-Ruiz, Belén Melián-Batista, and J Marcos Moreno-Vega. "A machine learning-based system for berth scheduling at bulk terminals". In: *Expert Systems with Applications* 87 (2017), pp. 170–182.

[Leó+17b] Alan Davila de León, Eduardo Lalla-Ruiz, Belén Melián-Batista, and J Marcos Moreno-Vega. "Meta-learning-based system for solving logistic optimization problems". In: *International Conference on Computer Aided Systems Theory*. Springer. 2017, pp. 339–346.

[Li+15] Jian Li, Panos M Pardalos, Hao Sun, Jun Pei, and Yong Zhang. "Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups". In: *Expert Systems with Applications* 42.7 (2015), pp. 3551–3561.

[Li+16] Chao Li, Xiaogeng Chu, Yingwu Chen, and Lining Xing. "A knowledge-based technique for initializing a genetic algorithm". In: *Journal of Intelligent & Fuzzy Systems* 31.2 (2016), pp. 1145–1152.

[LJCCC08] Antonio López Jaimes, Carlos A Coello Coello, and Debrup Chakraborty. "Objective reduction using a feature selection technique". In: *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. 2008, pp. 673–680.

[LJMN07] Francisco Chagas de Lima Junior, Jorge Dantas de Melo, and Adriao Duarte Doria Neto. "Using q-lea rning algorithm for initialization of the grasp metaheuristic and genetic algorithm". In: *2007 International Joint Conference on Neural Networks*. IEEE. 2007, pp. 1243–1248.

[LK73] Shen Lin and Brian W Kernighan. "An effective heuristic algorithm for the traveling-salesman problem". In: *Operations research* 21.2 (1973), pp. 498–516.

[LM04] Sushil J Louis and John McDonnell. "Learning with case-injected genetic algorithms". In: *IEEE Transactions on Evolutionary Computation* 8.4 (2004), pp. 316–328.

[LMR20] Andrea Lodi, Luca Mossina, and Emmanuel Rachelson. "Learning to handle parameter perturbations in combinatorial optimization: an application to facility location". In: *EURO Journal on Transportation and Logistics* (2020), p. 100023.

[LMS03] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. "Iterated local search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[LN87] Gilbert Laporte and Yves Nobert. "Exact algorithms for the vehicle routing problem". In: *North-Holland Mathematics Studies*. Vol. 132. Elsevier, 1987, pp. 147–184.

[LO05] Xiaonan Li and Sigurdur Olafsson. "Discovering dispatching rules using data mining". In: *Journal of Scheduling* 8.6 (2005), pp. 515–527.

[LSRVMG18] E López-Santana, W Rodríguez-Vásquez, and G Méndez-Giraldo. "A hybrid expert system, clustering and ant colony optimization approach for scheduling and routing problem in courier services". In: *International Journal of Industrial Engineering Computations* 9.3 (2018), pp. 369–396.

[LSS10] Ilya Loshchilov, Marc Schoenauer, and Michèle Sebag. "A mono surrogate for multiobjective optimization". In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 2010, pp. 471–478.

[LT12] Xin-Lan Liao and Chuan-Kang Ting. "Evolutionary algorithms using adaptive mutation for the selective pickup and delivery problem". In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–8.

[LT16] Kun Li and Huixin Tian. "A two-level self-adaptive variable neighborhood search algorithm for the prize-collecting vehicle routing problem". In: *Applied Soft Computing* 43 (2016), pp. 469–479.

[Luc+20] Flavien Lucas, Romain Billot, Marc Sevaux, and Kenneth Sörensen. "Reducing Space Search in Combinatorial Optimization Using Machine Learning Tools". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2020, pp. 143–150.

[Lug17] Edwin Lughofer. "On-line active learning: A new paradigm to improve practical useability of data stream modeling methods". In: *Information Sciences* 415 (2017), pp. 356–376.

[LZY19] Hao Lu, Xingwen Zhang, and Shuang Yang. "A Learning-based Iterative Method for Solving Vehicle Routing Problems". In: *International Conference on Learning Representations*. 2019.

[Mar+11] Simon Martin, Djamila Ouelhadj, Patrick Beullens, and Ender Ozcan. *A generic agent-based framework for cooperative search using pattern matching and reinforcement learning*. Tech. rep. 2011.

[Mar+16] Simon Martin, Djamila Ouelhadj, Patrick Beullens, Ender Ozcan, Angel A Juan, and Edmund K Burke. "A multi-agent based cooperative approach to scheduling and routing". In: *European Journal of Operational Research* 254.1 (2016), pp. 169–178.

[Mat+09] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, and Michèle Sebag. "Extreme compass and dynamic multi-armed bandits for adaptive operator selection". In: *2009 IEEE Congress on Evolutionary Computation*. IEEE. 2009, pp. 365–372.

[Mat+11] Jorge Maturana, Alvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, and Michele Sebag. "Adaptive operator selection and management in evolutionary algorithms". In: *Autonomous Search*. Springer, 2011, pp. 161–189.

[MCK08] David Meignan, Jean-Charles Créput, and Abderrafiâa Koukam. "A coalition-based metaheuristic for the vehicle routing problem". In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 1176–1182.

[MCK09] David Meignan, Jean-Charles Créput, and Abderrafiaa Koukam. "A cooperative and self-adaptive metaheuristic for the facility location problem". In: *Proceedings of the 11th annual conference on Genetic and evolutionary computation*. 2009, pp. 317–324.

[MDC14] Tommy Messelis and Patrick De Causmaecker. "An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem". In: *European Journal of Operational Research* 233.3 (2014), pp. 511–528.

[Mer+13] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, Jakob Bossek, and Frank Neumann. "A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem". In: *Annals of Mathematics and Artificial Intelligence* 69.2 (2013), pp. 151–182.

[Met+53] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[MGS20] Hadi Mosadegh, SMT Fatemi Ghomi, and Gürsel A Süer. "Stochastic mixed-model assembly line sequencing problem: Mathematical modeling and Q-learning based simulated annealing hyper-heuristics". In: *European Journal of Operational Research* 282.2 (2020), pp. 530–544.

[MH97] Nenad Mladenović and Pierre Hansen. "Variable neighborhood search". In: *Computers & operations research* 24.11 (1997), pp. 1097–1100.

[Mic00] Ryszard S Michalski. "Learnable evolution model: Evolutionary processes guided by machine learning". In: *Machine learning* 38.1-2 (2000), pp. 9–40.

[Mir+18] Enrico S Miranda, Fabio Fabris, Chrystian GM Nascimento, Alex A Freitas, and Alexandre CM Oliveira. "Meta-Learning for Recommending Metaheuristics for the MaxSAT Problem". In: *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE. 2018, pp. 169–174.

[MJL19] JN Min, C Jin, and LJ Lu. "Maximum-minimum distance clustering method for split-delivery vehicle-routing problem: Case studies and performance comparisons." In: *Advances in Production Engineering & Management* 14.1 (2019), pp. 125–135.

[MJTM19] Mehrdad Mohammadi, Payman Jula, and Reza Tavakkoli-Moghaddam. "Reliable single-allocation hub location problem with disruptions". In: *Transportation Research Part E: Logistics and Transportation Review* 123 (2019), pp. 90–120.

[MKC10] David Meignan, Abderrafiaa Koukam, and Jean-Charles Créput. "Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism". In: *Journal of Heuristics* 16.6 (2010), pp. 859–879.

[MKN20] Taha Mostafaie, Farzin Modarres Khiyabani, and Nima Jafari Navimipour. "A systematic study on meta-heuristic approaches for solving the graph coloring problem". In: *Computers & Operations Research* 120 (2020), p. 104850.

[MKY11] Alberto Moraglio, Yong-Hyuk Kim, and Yourim Yoon. "Geometric surrogate-based optimisation for permutation-based problems". In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. 2011, pp. 133–134.

[MLS10] Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion. "Autonomous operator management for evolutionary algorithms". In: *Journal of Heuristics* 16.6 (2010), pp. 881–909.

[MOF92] Olivier Martin, Steve W Otto, and Edward W Felten. "Large-step Markov chains for the TSP incorporating local search heuristics". In: *Operations Research Letters* 11.4 (1992), pp. 219–224.

[Moh+16] Mehrdad Mohammadi, Reza Tavakkoli-Moghaddam, Ali Siadat, and Jean-Yves Dantan. “Design of a reliable logistics network with hub disruption under uncertainty”. In: *Applied Mathematical Modelling* 40.9-10 (2016), pp. 5621–5642.

[Mor19] Behzad Moradi. “The new optimization algorithm for the vehicle routing problem with time windows using multi-objective discrete learnable evolution model”. In: *Soft Computing* (2019), pp. 1–29.

[Mos+89] Pablo Moscato et al. “On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms”. In: *Caltech concurrent computation program, C3P Report* 826 (1989), p. 1989.

[MR07] Jorge Maturana and María-Cristina Riff. “Solving the short-term electrical generation scheduling problem by an adaptive evolutionary approach”. In: *European Journal of Operational Research* 179.3 (2007), pp. 677–691.

[MS08] Jorge Maturana and Frédéric Saubion. “A compass to guide genetic algorithms”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 256–265.

[MYE18] Shoma Miki, Daisuke Yamamoto, and Hiroyuki Ebara. “Applying deep learning and reinforcement learning to traveling salesman problem”. In: *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE. 2018, pp. 65–70.

[Nar03] Alexander Nareyek. “Choosing search heuristics by non-stationary reinforcement learning”. In: *Metaheuristics: Computer decision-making*. Springer, 2003, pp. 523–544.

[Nas+19] Mohammad Mahdi Nasiri, Sadegh Salesi, Ali Rahbari, Navid Salmanzadeh Meydani, and Mojtaba Abdollai. “A data mining approach for population-based methods to solve the JSSP”. In: *Soft Computing* 23.21 (2019), pp. 11107–11122.

[NEJH83] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem”. In: *Omega* 11.1 (1983), pp. 91–95.

[Ngu+14] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. “Selection schemes in surrogate-assisted genetic programming for job shop scheduling”. In: *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer. 2014, pp. 656–667.

[OB14] Gabriela Ochoa and Edmund K Burke. “HyperILS: An Effective Iterated Local Search Hyper-heuristic for Combinatorial Optimisation”. In: *International Conference of the Practice and Theory of Automated Timetabling,(August)*. 2014, pp. 26–29.

[OL96] Ibrahim H. Osman and Gilbert Laporte. “Metaheuristics: A bibliography”. In: *Annals of Operations Research* 63.5 (1996), pp. 511–623.

[Oli+10] Adriano LI Oliveira, Petronio L Braga, Ricardo MF Lima, and Márcio L Cornélio. “GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation”. In: *information and Software Technology* 52.11 (2010), pp. 1155–1166.

[Özt+20] Hande Öztop, Mehmet Fatih Tasgetiren, Levent Kandiller, and Quan-Ke Pan. "A Novel General Variable Neighborhood Search through Q-Learning for No-Idle Flowshop Scheduling". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2020, pp. 1–8.

[Pag54] Ewan S Page. "Continuous inspection schemes". In: *Biometrika* 41.1/2 (1954), pp. 100–115.

[PBA13] Erik Pitzer, Andreas Beham, and Michael Affenzeller. "Automatic algorithm selection for the quadratic assignment problem using fitness landscape analysis". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2013, pp. 109–120.

[PDK18] Lucas Pavelski, Myriam Delgado, and Marie-Eleonore Kessaci. "Meta-learning for optimization: A case study on the flowshop problem using decision trees". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2018, pp. 1–8.

[PE02] James E Pettinger and Richard M Everson. "Controlling genetic algorithms with reinforcement learning". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. 2002, pp. 692–692.

[Pel+20] Julien Pelamatti, Loïc Brevault, Mathieu Balesdent, El-Ghazali Talbi, and Yannick Guerin. "Overview and Comparison of Gaussian Process-Based Surrogate Models for Mixed Continuous and Discrete Variables: Application on Aerospace Design Problems". In: *High-Performance Simulation-Based Optimization*. Springer, 2020, pp. 189–224.

[Pen+19] Bo Peng, Yuan Zhang, Yuvraj Gajpal, and Xiding Chen. "A Memetic Algorithm for the Green Vehicle Routing Problem". In: *Sustainability* 11.21 (2019), p. 6055.

[Pha+19] Han Duy Phan, Kirsten Ellis, Jan Carlo Barca, and Alan Dorin. "A survey of dynamic parameter setting methods for nature-inspired swarm intelligence algorithms". In: *Neural Computing and Applications* (2019), pp. 1–22.

[PK17] Junheung Park and Kyoung-Yun Kim. "Meta-modeling using generalized regression neural network and particle swarm optimization". In: *Applied Soft Computing* 51 (2017), pp. 354–369.

[PKD18] Lucas Marcondes Pavelski, Marie-Éléonore Kessaci, and Myriam Regattieri Delgado. "Recommending meta-heuristics and configurations for the flowshop problem via meta-learning: analysis and design". In: *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE. 2018, pp. 163–168.

[PS19] Federico Pagnozzi and Thomas Stützle. "Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems". In: *European Journal of Operational Research* 276.2 (2019), pp. 409–421.

[PSS08] Bhupendra Kumar Pathak, Sanjay Srivastava, and Kamal Srivastava. "Neural network embedded multiobjective genetic algorithm to solve non-linear time-cost tradeoff problems of project scheduling". In: *Journal of Scientific and Industrial Research* 67.2 (2008), pp. 124–131.

[PTL08] Quan-Ke Pan, Mehmet Fatih Tasgetiren, and Yun-Chia Liang. "A discrete differential evolution algorithm for the permutation flowshop scheduling problem". In: *Computers & Industrial Engineering* 55.4 (2008), pp. 795–816.

[Qas+13] Sultan Noman Qasem, Siti Mariyam Shamsuddin, Siti Zaiton Mohd Hashim, Maslina Darus, and Eiman Al-Shammari. "Memetic multiobjective particle swarm optimization-based radial basis function network for classification problems". In: *Information Sciences* 239 (2013), pp. 165–190.

[Ram+05] Iloneide CO Ramos, Marco César Goldbarg, Elizabeth G Goldbarg, and Adriao Duarte Dória Neto. "Logistic regression for parameter tuning on an evolutionary algorithm". In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 2. IEEE. 2005, pp. 1061–1068.

[Rei91] Gerhard Reinelt. "TSPLIBâĂŤA traveling salesman problem library". In: *ORSA journal on computing* 3.4 (1991), pp. 376–384.

[Ric+76] John R. Rice et al. "The algorithm selection problem". In: *Advances in computers* 15.65-118 (1976), p. 5.

[RN94] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Citeseer, 1994.

[RPM06] Marcos Henrique Ribeiro, Alexandre Plastino, and Simone L Martins. "Hybridization of GRASP metaheuristic with data mining techniques". In: *Journal of Mathematical Modelling and Algorithms* 5.1 (2006), pp. 23–41.

[RR+21] Felipe de la Rosa-Rivera, Jose I Nunez-Varela, José C Ortiz-Bayliss, and Hugo Terashima-Marín. "Algorithm selection for solving educational timetabling problems". In: *Expert Systems with Applications* 174 (2021), p. 114694.

[RS07] Rubén Ruiz and Thomas Stützle. "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem". In: *European Journal of Operational Research* 177.3 (2007), pp. 2033–2049.

[RTS08] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. "Opposition-based differential evolution". In: *IEEE Transactions on Evolutionary computation* 12.1 (2008), pp. 64–79.

[Sad+19] Souhila Sadeg, Leila Hamdad, Mouloud Haouas, Kouider Abderrahmane, Karima Benatchba, and Zineb Habbas. "Unsupervised Learning Bee Swarm Optimization Metaheuristic". In: *International Work-Conference on Artificial Neural Networks*. Springer. 2019, pp. 773–784.

[Sad+20] Souhila Sadeg, Leila Hamdad, Omar Kada, Karima Benatchba, and Zineb Habbas. "Meta-learning to Select the Best Metaheuristic for the MaxSAT Problem". In: *International Symposium on Modelling and Implementation of Complex Systems*. Springer. 2020, pp. 122–135.

[Sak+10] Yoshitaka Sakurai, Kouhei Takada, Takashi Kawabe, and Setsuo Tsuruta. "A method to control parameters of evolutionary algorithms by using reinforcement learning". In: *2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems*. IEEE. 2010, pp. 74–79.

[Sam88] Arthur L Samuel. “Some studies in machine learning using the game of checkers. IIâĂŤrecent progress”. In: *Computer Games I* (1988), pp. 366–400.

[San+06] Haroldo G Santos, Luiz Satoru Ochi, Euler Horta Marinho, and Lúcia Maria de A Drummond. “Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem”. In: *Neurocomputing* 70.1-3 (2006), pp. 70–77.

[San+10] Joao Paulo Queiroz dos Santos, Francisco Chagas de Lima Júnior, Rafael Marrocos Magalhães, Jorge Dantas de Melo, and Adrião Duarte Doria Neto. “A Parallel Hybrid Implementation Using Genetic Algorithms, GRASP and Reinforcement Learning for the Salesman Traveling Problem”. In: *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, pp. 345–369.

[San+14] João Paulo Queiroz dos Santos, Jorge Dantas de Melo, Adrião Dória Duarte Neto, and Daniel Aloise. “Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search”. In: *Expert Systems with Applications* 41.10 (2014), pp. 4939–4949.

[Sax+12] Dhish Kumar Saxena, João A Duro, Ashutosh Tiwari, Kalyanmoy Deb, and Qingfu Zhang. “Objective reduction in many-objective optimization: Linear and nonlinear algorithms”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (2012), pp. 77–99.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Seg+16] Eduardo Segredo, Ben Paechter, Emma Hart, and Carlos Ignacio González-Vila. “Hybrid parameter control approach applied to a diversity-based multi-objective memetic algorithm for frequency assignment problems”. In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2016, pp. 1517–1524.

[SFH16] Armando M Leite da Silva, Muriell R Freire, and Leonardo M Honório. “Transmission expansion planning optimization by adaptive multi-operator evolutionary algorithms”. In: *Electric Power Systems Research* 133 (2016), pp. 173–181.

[SG13] Kenneth Sörensen and Fred Glover. “Metaheuristics”. In: *Encyclopedia of operations research and management science* 62 (2013), pp. 960–970.

[Sgh+15] Ines Sghir, Jin-Kao Hao, Ines Ben Jaafar, and Khaled Ghédira. “A multi-agent based optimization method applied to the quadratic assignment problem”. In: *Expert Systems with Applications* 42.23 (2015), pp. 9252–9262.

[Sha98] Paul Shaw. “Using constraint programming and local search methods to solve vehicle routing problems”. In: *International conference on principles and practice of constraint programming*. Springer. 1998, pp. 417–431.

[Sil+15] Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza, and Sabrina Moreira de Oliveira. “A multi-agent metaheuristic optimization framework with cooperation”. In: *2015 Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE. 2015, pp. 104–109.

[Sil+18] Maria Amélia Lopes Silva, Sergio Ricardo de Souza, Marcone Jamilson Freitas Souza, and Moacir Felizardo de Franca Filho. “Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis”. In: *Applied Soft Computing* 71 (2018), pp. 433–459.

[Sil+19] Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza, and Ana Lúcia C Bazzan. “A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems”. In: *Expert Systems with Applications* 131 (2019), pp. 148–171.

[SJG18] Ines Sghir, Ines Ben Jaafar, and Khaled Ghédira. “A Multi-Agent based Optimization Method for Combinatorial Optimization Problems”. In: *International Journal on Artificial Intelligence Tools* 27.05 (2018), p. 1850021.

[SM08] Kate A Smith-Miles. “Towards insightful algorithm selection for optimisation using meta-learning concepts”. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. ieee. 2008, pp. 4118–4124.

[SM+14] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. “Towards objective measures of algorithm performance across instance space”. In: *Computers & Operations Research* 45 (2014), pp. 12–24.

[SMHL10] Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. “Understanding TSP difficulty by learning from evolved instances”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2010, pp. 266–280.

[SML12] Kate Smith-Miles and Leo Lopes. “Measuring instance difficulty for combinatorial optimization problems”. In: *Computers & Operations Research* 39.5 (2012), pp. 875–889.

[SP97] Rainer Storn and Kenneth Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[SR10] L Shi and K Rasheed. “A survey of fitness approximation methods applied in evolutionary algorithms”. In: *Computational intelligence in expensive optimization problems*. Springer, 2010, pp. 3–28.

[SRS10] Hemant Kumar Singh, Tapabrata Ray, and Warren Smith. “Surrogate assisted simulated annealing (SASA) for constrained multi-objective optimization”. In: *IEEE congress on evolutionary computation*. IEEE. 2010, pp. 1–8.

[ST12] Yoshitaka Sakurai and Setsuo Tsuruta. “A population based rewarding for Reinforcement Learning to control Genetic Algorithms”. In: *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*. IEEE. 2012, pp. 686–691.

[STÃ19] Heda Song, Isaac Triguero, and Ender ÃŰzcan. “A review on the self and dual interactions between machine learning and optimisation”. In: *Progress in Artificial Intelligence* 8.2 (2019), pp. 143–165.

[Sun+19] Jianyong Sun, Hu Zhang, Aimin Zhou, Qingfu Zhang, and Ke Zhang. “A new learning-based adaptive multi-objective evolutionary algorithm”. In: *Swarm and evolutionary computation* 44 (2019), pp. 304–319.

[Tai90] Eric Taillard. “Some efficient heuristic methods for the flow shop sequencing problem”. In: *European journal of Operational research* 47.1 (1990), pp. 65–74.

[Tai93] Eric Taillard. “Benchmarks for basic scheduling problems”. In: *european journal of operational research* 64.2 (1993), pp. 278–285.

[Tal02] E-G Talbi. “A taxonomy of hybrid metaheuristics”. In: *Journal of heuristics* 8.5 (2002), pp. 541–564.

[Tal09] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, 2009.

[Tal16] El-Ghazali Talbi. “Combining metaheuristics with mathematical programming, constraint programming and machine learning”. In: *Annals of Operations Research* 240.1 (2016), pp. 171–215.

[Tal20] El-Ghazali Talbi. “Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics”. In: *Working paper* (2020), pp. 1–30.

[TSH20] Renata Turkeš, Kenneth SÃűrensen, and Lars Magnus Hvattum. “Meta-analysis of Metaheuristics: Quantifying the Effect of Adaptiveness in Adaptive Large Neighborhood Search”. In: *European Journal of Operational Research* (2020). DOI: 10.1016/j.ejor.2020.10.045.

[TSH21] Renata Turkeš, Kenneth Sörensen, and Lars Magnus Hvattum. “Meta-analysis of metaheuristics: quantifying the effect of adaptiveness in adaptive large neighborhood search”. In: *European Journal of Operational Research* 292.2 (2021), pp. 423–442.

[Tsp] “www.elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html”. In: ().

[TZ19] Simon Thevenin and Nicolas Zufferey. “Learning Variable Neighborhood Search for a scheduling problem with time windows and rejections”. In: *Discrete Applied Mathematics* 261 (2019), pp. 344–353.

[VLMT11] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. “GPU computing for parallel local search metaheuristic algorithms”. In: *IEEE transactions on computers* 62.1 (2011), pp. 173–185.

[VRF15] Eva Vallada, Rubén Ruiz, and Jose M Framinan. “New hard benchmark for flowshop scheduling problems minimising makespan”. In: *European Journal of Operational Research* 240.3 (2015), pp. 666–677.

[VT03] Christos Voudouris and Edward PK Tsang. “Guided local search”. In: *Handbook of metaheuristics*. Springer, 2003, pp. 185–218.

[VT99] Christos Voudouris and Edward Tsang. “Guided local search and its application to the traveling salesman problem”. In: *European journal of operational research* 113.2 (1999), pp. 469–499.

[WA05] Stefan Wagner and Michael Affenzeller. “Heuristiclab: A generic and extensible optimization environment”. In: *Adaptive and Natural Computing Algorithms*. Springer, 2005, pp. 538–541.

[Wal+12] James D Walker, Gabriela Ochoa, Michel Gendreau, and Edmund K Burke. “Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework”. In: *International conference on learning and intelligent optimization*. Springer. 2012, pp. 265–276.

[Wan+20a] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. "Generalizing from a few examples: A survey on few-shot learning". In: *ACM Computing Surveys (CSUR)* 53.3 (2020), pp. 1–34.

[Wan+20b] Yiyuan Wang, Shiwei Pan, Chenxi Li, and Minghao Yin. "A local search algorithm with reinforcement learning based repair procedure for minimum weight independent dominating set". In: *Information Sciences* 512 (2020), pp. 533–548.

[Wat89] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[Wau+13] Tony Wauters, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. "Boosting metaheuristic search using reinforcement learning". In: *Hybrid Metaheuristics*. Springer, 2013, pp. 433–452.

[WBK20] Jeremy Watt, Reza Borhani, and Aggelos K Katsaggelos. *Machine learning refined: Foundations, algorithms, and applications*. Cambridge University Press, 2020.

[WDS20] Jakub Wawrzyniak, Maciej Drozdowski, and Éric Sanlaville. "Selecting algorithms for large berth allocation problems". In: *European Journal of Operational Research* 283.3 (2020), pp. 844–862.

[Wil92] Frank Wilcoxon. "Individual comparisons by ranking methods". In: *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[Wit+05] Ian H Witten, Eibe Frank, Mark A Hall, Christopher J Pal, and MINING DATA. "Practical machine learning tools and techniques". In: *DATA MINING*. Vol. 2. 2005, p. 4.

[WJ18] Handing Wang and Yaochu Jin. "A random forest-assisted evolutionary algorithm for data-driven constrained multiobjective combinatorial optimization of trauma systems". In: *IEEE transactions on cybernetics* 50.2 (2018), pp. 536–549.

[WM97] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[Woe03] Gerhard J Woeginger. "Exact algorithms for NP-hard problems: A survey". In: *Combinatorial optimizationâĂŤeureka, you shrink!* Springer, 2003, pp. 185–207.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[WT17a] Xianpeng Wang and Lixin Tang. "A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem". In: *Computers & Operations Research* 79 (2017), pp. 60–77.

[WT17b] Wenguo Wu and Shih-Pang Tseng. "An Improved Learnable Evolution Model for Discrete Optimization Problem". In: *Advances in Intelligent Information Hiding and Multimedia Signal Processing*. Springer, 2017, pp. 333–340.

[WWH11] Janusz Wojtusiak, Tobias Warden, and Otthein Herzog. "Agent-based pickup and delivery planning: the learnable evolution model approach". In: *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE. 2011, pp. 1–8.

[WWH12] Janusz Wojtusiak, Tobias Warden, and Otthein Herzog. “The learnable evolution model in agent-based delivery optimization”. In: *Memetic Computing* 4.3 (2012), pp. 165–181.

[Xia+20] Xiaoshu Xiang, Ye Tian, Jianhua Xiao, and Xingyi Zhang. “A Clustering-Based Surrogate-Assisted Multi-objective Evolutionary Algorithm for Shelter Location under Uncertainty of Road Networks”. In: *IEEE Transactions on Industrial Informatics* 16.12 (2020), pp. 7544–7555.

[XP16] Ting Xiang and Dazhi Pan. “Clustering algorithm for split delivery vehicle routing problem”. In: *Journal of Computer Applications* 36.11 (2016), pp. 3141–3145.

[Xue+15] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. “A survey on evolutionary computation approaches to feature selection”. In: *IEEE Transactions on Evolutionary Computation* 20.4 (2015), pp. 606–626.

[YA01] Tankut Yalcinoz and Halis Altun. “Power economic dispatch using a hybrid genetic algorithm”. In: *IEEE power engineering review* 21.3 (2001), pp. 59–60.

[Yu+17] Haibo Yu, Ying Tan, Chaoli Sun, and Jianchao Zeng. “Clustering-based evolution control for surrogate-assisted particle swarm optimization”. In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2017, pp. 503–508.

[Yua+14] Zhi Yuan, Stephanus Daniel Handoko, Duc Thien Nguyen, and Hoong Chuin Lau. “An empirical study of off-line configuration and on-line adaptation in operator selection”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2014, pp. 62–76.

[Zan+09] M Zandieh, M Amiri, B Vahdani, and R Soltani. “A robust parameter design for multi-response problems”. In: *Journal of computational and applied mathematics* 230.2 (2009), pp. 463–476.

[ZEC10] M Zennaki and A Ech-Cherif. “A new machine learning based approach for tuning metaheuristics for the solution of hard combinatorial optimization problems”. In: *Journal of Applied Sciences(Faisalabad)* 10.18 (2010), pp. 1991–2000.

[ZFX19] Yuan Zheng, Xiaogang Fu, and Yanwen Xuan. “Data-Driven Optimization Based on Random Forest Surrogate”. In: *2019 6th International Conference on Systems and Informatics (ICSAI)*. IEEE. 2019, pp. 487–491.

[ZG09] Xiaojin Zhu and Andrew B Goldberg. “Introduction to semi-supervised learning”. In: *Synthesis lectures on artificial intelligence and machine learning* 3.1 (2009), pp. 1–130.

[Zha+11] Jun Zhang, Zhi-hui Zhan, Ying Lin, Ni Chen, Yue-jiao Gong, Jing-hui Zhong, Henry SH Chung, Yun Li, and Yu-hui Shi. “Evolutionary computation meets machine learning: A survey”. In: *IEEE Computational Intelligence Magazine* 6.4 (2011), pp. 68–75.

[Zha+16a] Mohammad Zhalechian, Reza Tavakkoli-Moghaddam, Behzad Zahiri, and Mehrdad Mohammadi. “Sustainable design of a closed-loop location-routing-inventory supply chain network under mixed uncertainty”. In: *Transportation Research Part E: Logistics and Transportation Review* 89 (2016), pp. 182–214.

[Zha+16b] Hu Zhang, Aimin Zhou, Shenmin Song, Qingfu Zhang, Xiao-Zhi Gao, and Jun Zhang. "A self-organizing multiobjective evolutionary algorithm". In: *IEEE Transactions on Evolutionary Computation* 20.5 (2016), pp. 792–806.

[Zha17] Jiashan Zhang. "An efficient density-based clustering algorithm for the capacitated vehicle routing problem". In: *2017 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. IEEE. 2017, pp. 465–469.

[Zha+21] Fuqing Zhao, Lixin Zhang, Jie Cao, and Jianxin Tang. "A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem". In: *Computers & Industrial Engineering* 153 (2021), p. 107082.

[ZHD16] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. "Reinforcement learning based local search for grouping problems: A case study on graph coloring". In: *Expert Systems with Applications* 64 (2016), pp. 412–422.

[ZHD20] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. "Frequent Pattern-Based Search: A Case Study on the Quadratic Assignment Problem". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2020).

[Zhe15] Yu-Jun Zheng. "Water wave optimization: a new nature-inspired metaheuristic". In: *Computers & Operations Research* 55 (2015), pp. 1–11.

[Zho+05] Zongzhao Zhou, Yew Soon Ong, My Hanh Nguyen, and Dudy Lim. "A study on polynomial regression and Gaussian process global surrogate model in hierarchical surrogate-assisted evolutionary algorithm". In: *2005 IEEE congress on evolutionary computation*. Vol. 3. IEEE. 2005, pp. 2832–2839.

Acronyms

ABC Artificial Bee Colony. 37
ACO Ant Colony Optimization. 37
ANN Artificial Neural Network. 40
AOS Adaptive Operator Selection. 31
ARs Association Rules. 40

BLS Breakout Local Search. 37

COP Combinatorial Optimization Problem. 28

DE Differential Evolution. 37
DLS Descent-based Local Search. 37
DQL Deep Q-learning. 43
DR Dimension Reduction. 40
DRL Deep Reinforcement Learning. 43
DT Decision Tree. 40

EAs Evolutionary Algorithms. 37

FSL Few-Shot Learning. 40

GA Genetic Algorithm. 37
GB Gradient Boosting. 40
GD Great Deluge. 37
GH Greedy Heuristic. 37
GLS Guided Local Search. 37
GRASP Greedy Randomized Adaptive Search Procedure. 37
GRH Greedy Randomized Heuristic. 37

HC Hill Climbing. 37
HS Harmony Search. 37

ICA Imperialist Competitive Algorithm. 37
ILS Iterated Local Search. 32

kNN k-Nearest Neighbor. 40

LA Learning Automata. 42
LDA Linear Discriminant Analysis. 40

LNS Large Neighborhood Search. 37

LogR Logistic Regression. 40

LR Linear Regression. 40

LSL Low-Shot Learning. 40

MA Memetic Algorithm. 37

MCA Multiple Correspondence Analysis. 40

MDP Markov decision process. 42

MH Meta-heuristic. 29

ML Machine Learning. 29

NB Naive Bayes. 40

NN Nearest Neighbor. 37

OPRL Opposition-based Reinforcement Learning. 42

PCA Principal Component Analysis. 40

PFSP Permutation Flowshop Scheduling Problem. 32

PSO Particle Swarm Optimization. 37

QL Q-learning. 42

RF Random Forest. 40

RL Reinforcement Learning. 31

RSM Response Surface Methodology. 102

SA Simulated Annealing. 37

SARSA State-action-reward-state-action. 42

SNNC Shared Nearest Neighbor Clustering. 40

SOM Self-Organizing Map. 40

SVM Support Vector Machine. 40

TPD Transition Probability Distribution. 42

TS Tabu Search. 37

TSP Traveling Salesman Problem. 28

VNS Variable Neighborhood Search. 37

WWO Water Wave Optimization. 37

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Hybridation des métaheuristiques avec l'apprentissage automatique pour l'optimisation combinatoire : une taxonomie et apprendre à sélectionner des opérateurs

Mots clés : Optimisation combinatoire, méta-heuristiques, apprentissage automatique, sélection adaptative des opérateurs

Résumé : Cette thèse intègre des techniques d'apprentissage automatique dans des méta-heuristiques pour résoudre des problèmes d'optimisation combinatoire. Cette intégration guidera les méta-heuristiques vers la prise de meilleures décisions et par conséquent à rendre les méta-heuristiques plus efficaces. Cette thèse, tout d'abord, fournit une revue complète mais technique de la littérature et propose une taxonomie unifiée sur les différentes manières d'intégration. Pour chaque type d'intégration, une analyse et une discussion complètes sont fournies sur les détails techniques, y compris les défis, les avantages, les inconvénients et les perspectives.

Nous nous concentrons ensuite sur une intégration particulière et abordons le problème de la sélection adaptative des opérateurs dans les méta-heuristiques utilisant des techniques d'apprentissage par renforcement.

Plus précisément, nous proposons un cadre général qui intègre l'algorithme Q-learning dans l'algorithme de recherche locale itérée afin de sélectionner de manière adaptative les opérateurs de recherche les plus appropriés à chaque étape du processus de recherche en fonction de leur historique de performance.

Le cadre proposé est appliqué à deux problèmes d'optimisation combinatoire, le problème du voyageur de commerce et le problème d'ordonnancement de type flowshop de permutation. Dans les deux applications, le cadre proposé est plus performant en termes de qualité de solution et de taux de convergence qu'une sélection aléatoire d'opérateurs. De plus, nous observons que le cadre proposé montre un comportement d'état de l'art lors de la résolution du problème d'ordonnancement des flux de permutation.

Title: Hybridizing Metaheuristics with Machine Learning for Combinatorial Optimization: A Taxonomy and Learning to Select Operators

Keywords: Combinatorial optimization, meta-heuristiques, machine learning, adaptive operator selection

Abstract: This thesis integrates machine learning techniques into meta-heuristics for solving combinatorial optimization problems. This integration aims to guide the meta-heuristics toward making better decisions and consequently make meta-heuristics more efficient and improve their performance in terms of solution quality and convergence rate. This thesis, first, provides a comprehensive yet technical review of the literature and proposes a unified taxonomy on different ways of the integration. For each type of integration, a complete analysis and discussion is provided on technical details, including challenges, advantages, disadvantages, and perspectives.

From a technical aspect, we then focus on a particular integration and address the problem of adaptive operator selection in meta-heuristics using reinforcement learning techniques.

More precisely, we propose a general framework that integrates the Q-learning algorithm, as a reinforcement learning algorithm, into the iterated local search algorithm to adaptively and dynamically select the most appropriate search operators at each step of the search process based on their history of performance.

The proposed framework is finally applied on two combinatorial optimization problems, traveling salesman problem and permutation flowshop scheduling problem. In both applications, the framework better performance in terms of solution quality and convergence rate compared to a random selection of operators, especially for large size instances of the problems. Moreover, we observe that the proposed framework shows the state-of-the-art behavior when solving the permutation flowshop scheduling problem.