Exact and Hyper-heuristic Methods for Solving the Conference Scheduling Problem

Supplementary Materials: This document provides background information and additional materials related to this paper. It forms part of the PhD thesis prepared by Yaroslav

Yaroslav Pylyavskyy*, Ahmed Kheiri‡, Peter Jacko*†
*Lancaster University, Department of Management Science, Lancaster LA1 4YX, UK
‡Alliance Manchester Business School, The University of Manchester, Manchester UK
{y.pylyavskyy1, p.jacko}@lancaster.ac.uk
ahmed.kheiri@manchester.ac.uk
†Berry Consultants, 5 East Saint Helen Street, Abingdon OX14 5EG, UK

I. Integer Programming

Prior to describing the concept of integer programming, we first need to introduce the classical linear programming. Linear programming is defined as an optimisation method used to obtain the best solution in a mathematical model whose formulation is expressed in linear relationships. In other words, linear programming is an optimisation technique for minimisation or maximisation of a linear objective function, subject to linear equality and inequality constraints. Its feasible region is given by the set of all possible points of an optimisation problem that satisfy the constraints of the problem (e.g., see Fig. 1) [1], [2].

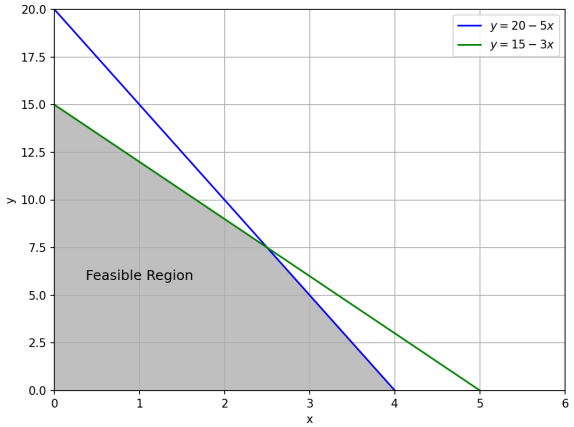

Fig. 1. Feasible Region of a Linear Programming Problem

A formulation example of a linear programming problem is as follows:

$$\begin{aligned} \min\ & c^\top x \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where x are continuous variables, c and b are given vectors, and A is a given matrix. In this example, the goal is to minimise the objective function, $c^T x$, with respect to the constraints, $Ax \leq b$ and $x \geq 0$.

Integer programming is usually considered as an extension of linear programming where some or all of its variables are restricted to take integer values. A model is defined as pure integer programming when all variables must be integers, whereas a mixed integer programming is a model where only some of the variables are restricted to integer variables. In contrast to linear programming problems, integer programming problems have feasible integer points instead of a feasible region (see Fig. 2). It should also be noted that the computational complexity of integer programming models is greater than linear programming models because of integer variables and, thus, much more difficult to solve. Due to this increased complexity, it is usually impractical to solve large scale real-world problems exactly with integer programming. Therefore, heuristics are often preferred as an alternative optimisation method [1], [3]–[5].

The formulation of an integer programming problem is similar to the linear programming problem. For instance, we can rewrite the previous formulation example to a pure integer programming problem as follows:

$$\begin{aligned} \min\ & c^\top x \\ & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z} \end{aligned}$$

where the only difference, comparing to the previous example, is that x variables are restricted to integer values.

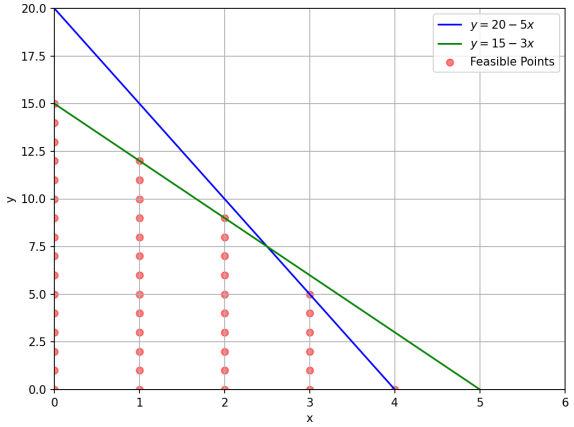

Fig. 2. Feasible Integer Points of an Integer Programming Problem

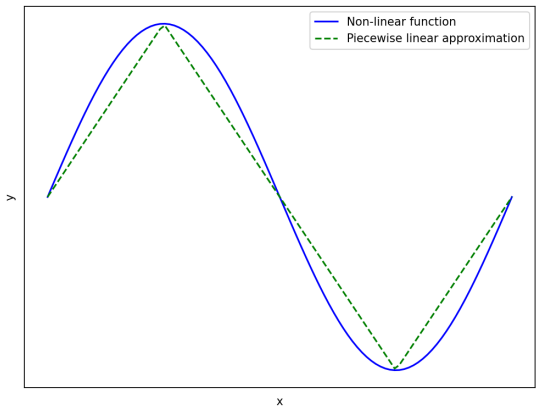

Fig. 3. A piecewise linear approximation to a non-linear function

A very practical and useful variant of integer programming is the zero-one linear programming (or binary integer programming) that is used for problems where all variables must be binary (either 0 or 1). The 0-1 variables are widely used to formulate problems in which decisions are discrete by nature (Yes/No decisions), and to facilitate the formulation of logical decisions and statements, e.g., if an ingredient i is used then a proportion of a_i must be respected. Moreover, 0-1 variables can be used to convert non-linear terms into linear such as the product of two 0-1 variables. For instance, assume we want to replace the product $\alpha_1\alpha_2$ with a new 0-1 variable β. We can achieve this by introducing the following constraints:

$$\begin{aligned}\beta &\geq \alpha_1 + \alpha_2 - 1\\ \beta &\leq \alpha_1\\ \beta &\leq \alpha_2\end{aligned}$$

These constraints enforce $\beta = 1$ only when $\alpha_1 = \alpha_2 = 1$, otherwise β is restricted to 0 if α_1, or α_2, or both are 0. Another similar use of 0-1 variables is the approximation of non-linear models as shown in Fig. 3. An integer programming model can be used to convert a certain type of non-linearity into linear terms at a cost in the size of the converted model though [1].

Due to these practical properties of the zero-one variables, binary integer programming models have been extensively used in numerous applications and problems such as scheduling [6], [7], the travelling salesman problem [8], [9], the knapsack problem [10], [11], packing and partitioning problems [12]–[14], the facility location problem [15], [16], and many others ranging from networks and graphs to capital investment problems [17]–[19].

A. Relaxations

Relaxation is a well-known modelling strategy within the field of mathematical optimisation to approximate difficult problems by similar problems that are usually easier to solve. The solution of a relaxed problem provides information about the original problem. For instance, removing integrality constraints from an integer programming problem results in a relaxed linear programming problem.

Another relaxation method is the Lagrangian relaxation which allows inequality constraints to be violated. Relaxed constraints are moved to the objective function and are multiplied with a Lagrange multiplier. As a result, a cost is incurred for each violation of the relaxed constraints. For example, assume we are given the following linear programming model:

$$\begin{aligned}\min\ & c^\top x\\ Ax &\leq b\\ x &\geq 0\end{aligned}$$

To obtain the Lagrangian relaxation of the problem, constraint $Ax \leq b$ is moved to the objective function and multiplied with a Lagrange multiplier $\lambda^\top$ as follows:

$$\begin{aligned}\min\ & c^\top x + \lambda^\top (Ax - b)\\ x &\geq 0\end{aligned}$$

where $\lambda^\top \geq 0$. Thus, an approximation of the original problem is achieved in which the objective function is penalised if the relaxed constraint gets violated [20].

A similar relaxation method to Lagrangian relaxation is the one introduced in [21]. However, the difference is that in his method the relaxed constraints are not moved to the objective function. Instead, new variables are introduced in both the objective function and the constraints. These variables are used to represent the exceeded violation amount of a constraint and the unused amount of a constraint. The work in [21] described this method as a mixture of a conventional objective function and a goal programming objective function which can

be formulated as follows:

$$\begin{aligned} \min\ & c^\top x + c^{'\top} u + c^{''\top} v \\ & Ax + u - v = b \\ & x \geq 0 \end{aligned}$$

where u indicates the unused amount of the constraint, v indicates the exceeded amount of the constraint, $c^{'\top}$ represents the cost of not fully utilising the constraint, and $c^{''\top}$ represents the cost of exceeding the constraint.

Apart from approximating difficult problems, relaxation methods have several additional benefits and uses which are described next. They can be used to obtain bounds in branch-and-bound algorithms for integer programming (e.g., [22]) and to overcome infeasibility issues in mathematical modelling. Additionally, they enable the handling of complex and soft constraints, and allow the expansion of the solution space by setting different weights values. That is, they allow the exploration of multiple solutions along with trade-offs which is beneficial for decision-makers as they are presented with more options. However, relaxation methods have certain drawbacks as well. They require the judgement of decision-makers upon selecting the right solution and the relaxed problem may not converge to the optimal solution of the original problem.

II. Hyper-heuristics

Sometimes, it is impractical to solve a problem exactly to obtain the optimal solution for various reasons such as computational complexity, size of the problem, and complexities involved in formulating the problem. In such cases, an effective alternative method is the implementation of heuristic techniques which sacrifice optimality to find near-optimal solutions within a short amount of time.

A well-known heuristic method to solve computationally hard optimisation problems of combinatorial nature is the local search algorithm, or also known as neighbourhood search. Typically, a combinatorial optimisation problem involves an objective function to minimise, or maximise, and a set of feasible, or candidate, solutions. The key mechanism of the local search algorithm is based on a neighbourhood structure, where a neighbourhood is defined as a set of solutions that are slightly different compared to a given solution. Local search algorithms apply local changes to explore neighbourhood solutions within the space of candidate solutions, or the search space, until a termination criterion is met [23], [24]. For instance, in a travelling salesman problem, a neighbourhood solution is a tour in which the visiting order of two cities is different. A significant part of heuristic methods is the criterion used to accept or reject a new solution, which is known as the move acceptance criterion. A common move acceptance criterion is the acceptance of the new solution only when it is better than the previous solution [25]. While local search algorithms are effective in terms of execution time and solution quality, they do have a pitfall. That is, they can stuck at a local optima when the neighbourhood search space is poorly structured (e.g., see Fig. 4).

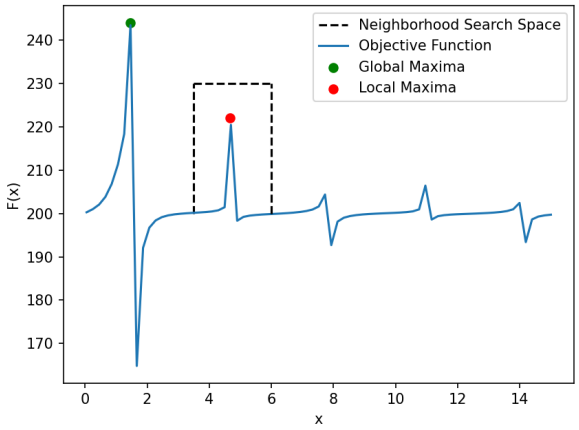

Fig. 4. Local search algorithm stuck at local maxima

Several methods exist to overcome this pitfall such as the usage of a random restart approach in which the local search algorithm is applied to multiple random initial solutions. Hence, this approach increases significantly the search space and the probability of finding a better local optima [26]. Another method is the implementation of the ruin and recreate principle where parts of the solution are destroyed and rebuilt or recreated afterwards [27]. Alternatively, more sophisticated algorithms may be used instead such as simulated annealing, genetic algorithms, ant colony optimisation, or hyper-heuristics which are described next.

Many combinatorial optimisation problems have been solved by means of hyper-heuristics which can be simply defined as a heuristic to choose heuristics. While hyper-heuristics were conceptualised in 1960s, they were formally defined in the early 2000s as a high-level automated search methodology for solving computationally hard problems by exploring the search space of low-level heuristics. Hyper-heuristics are classified into the following two categories: 1) *generation* hyper-heuristics that generate new heuristics, and 2) *selection* hyper-heuristics that select and apply a heuristic from a set of low-level heuristics. In addition, selection hyper-heuristics are further divided into: 1) *constructive* hyper-heuristics that build a solution from scratch, and 2) *perturbative* hyper-heuristics that iteratively improve a given complete solution [28]. In this work, we focus on selection perturbative hyper-heuristics to which we will simply refer as hyper-heuristics onwards. A typical framework of a hyper-heuristic usually includes two sequential steps, the heuristic selection and the move acceptance (see Fig. 5). While the former step represents a strategy for selecting and applying a low-level heuristic, the latter step represents a strategy regarding the acceptance or rejection of the newly created solution [29].

Hyper-heuristics have become popular due to their advantages over other customised methods. Their major benefit is that they are not limited to a single problem domain or a narrow class of problem instances, instead they are problem

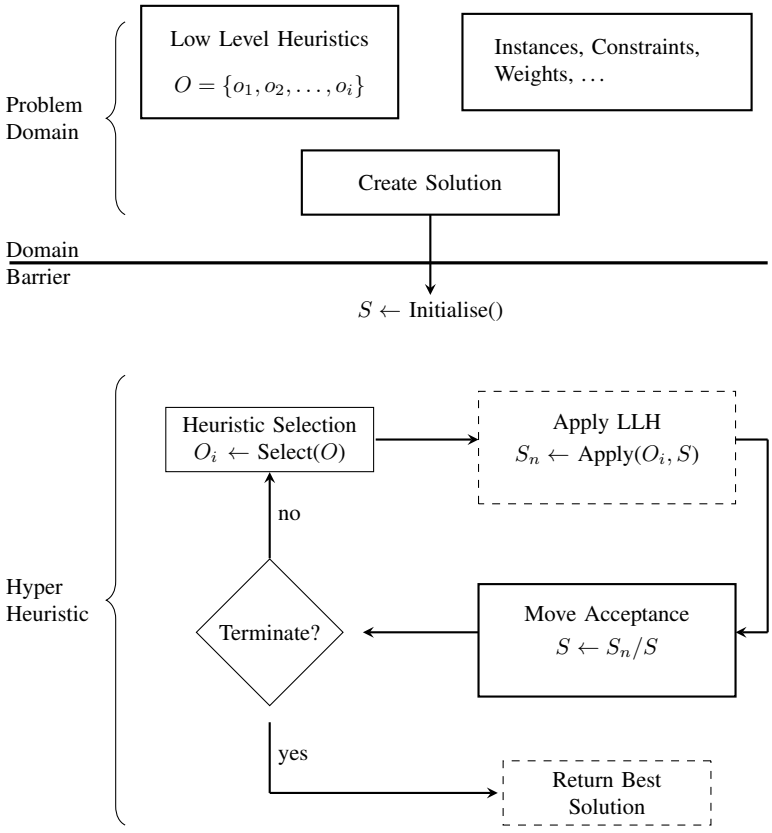

Fig. 5. A flowchart of the hyper-heuristic framework

independent and are applicable to a wide range of problem instances [28]–[30]. Additionally, hyper-heuristics are capable of learning by receiving feedback regarding the performance of low-level heuristics during the optimisation process. Two types of learning processes are identified in the literature: the *online* learning in which the algorithm directly learns while solving the problem, and the *offline* learning in which the algorithm learns by solving a set of training instances. Some recent studies, though, have incorporated a mixture of online and offline learning [28], [31]. Furthermore, it is relatively easy to implement hyper-heuristics, compared to other customised methods, as no information is required about the functionality of low-level heuristics. They only require the number or content of the low-level heuristics and the type of optimisation (minimisation or maximisation) [32], [33]. Last but not least, their purpose is to find an effective method of solving a problem by exploring the space of low-level heuristics. This is in contrast to other metaheuristic algorithms which are limited to finding a good solution by exploring directly the space of solutions [32], [34].

A. Heuristic Selection Strategies

In this step of the optimisation process, a low-level heuristic is selected from a set of low-level heuristics and is applied to the solution iteratively. Note that a low-level heuristic can be a simple operator, a metaheuristic, or even a hyper-heuristic. The performance of the algorithm depends heavily on the selected strategy as some low-level heuristics may perform better than others at certain stages of the optimisation process. For instance, low-level heuristics that perform major changes to the solution are usually more successful at early stages of optimisation rather than later stages. In addition, the sequence in which low-level heuristics are applied plays a crucial role in terms of performance too, because different solutions are obtained with altered sequences [32]. Heuristic selection strategies are defined as *deterministic* when a selection sequence is predefined, or *stochastic* when the selection depends on a probability distribution or learning mechanism [28], [29]. Some examples of heuristic selection strategies are presented next:

- Simple random: A low-level heuristic is randomly selected.
- Greedy: All low-level heuristics are applied and the best among them is selected at each iteration.
- Reinforcement learning: Each low-level heuristic is assigned an initial minimum score which increases or decreases based on performance. At each iteration, the low-level heuristic with the highest score is selected [32].
- Tabu search: Low-level heuristics are ranked similarly to reinforcement learning strategy. Upon solution improvement the rank is increased, otherwise the rank is decreased and the low-level heuristic enters the tabu list until the solution at hand changes. This strategy selects

the low-level heuristic with the highest rank that is not included in the tabu list [34].
- Sequence-based selection: In this strategy, low-level heuristics are selected based on the sequences of low-level heuristics with the best performance [30].

B. Move Acceptance Strategies

At this stage, the newly obtained solution is evaluated and is either accepted or rejected based on the selected strategy. Similarly to heuristic selection strategies, move acceptance strategies can be defined as *deterministic* or *stochastic*. Deterministic strategies usually accept non-worsening solutions, while stochastic strategies may accept or reject a solution according to some probabilistic criterion. Stochastic strategies may accept worse solutions during optimisation which is particularly useful for escaping local optima solutions and thus diversify the search space [28], [29]. A few examples of move acceptance strategies are the following:

- Only improving: The solution is only accepted if it is better than the previous.
- Improving and equal: All non-worsening solutions are accepted.
- Simulated annealing: Solutions are accepted based on the following probabilistic function:

$$p_t = e^{-\frac{\Delta f}{\Delta F(1-\frac{t}{T})}}$$

where Δf is the difference in the objective value at iteration t, T is the maximum number of iterations, and ΔF is the range of the maximum change in the objective value after a low-level heuristic is applied [35].
- Record-to-record travel: A worse solution is only accepted if it is not significantly exceeding the objective value of the previous solution [36].
- Late acceptance: Objective values are stored in a set of size L, and a new solution is only accepted if it is better than the L^{th} solution [37].

III. Matheuristics

The term matheuristic, as the name suggests, refers to an optimisation method in which ideas and methods are combined from both mathematical programming and heuristic techniques. [38] provided a general classification of matheuristics based on the nature of combination. They classified matheuristics into two main categories, namely the *collaborative combinations* and the *integrative combinations*. The former category refers to matheuristics in which exact and heuristic algorithms are separate parts and exchange information by being executed sequentially, intertwined or in parallel, while the latter category refers to matheuristics in which one technique is an embedded component of another technique. While [38] touched on a generic classification of matheuristics, additional classifications are defined in [39] which are discussed next:

- *Decomposition approaches:* In this approach, the master problem is decomposed into smaller and simpler sub-problems, where each sub-problem is solved by a specific solution method. Mathematical programming models are then used to solve some or all sub-problems to optimality or sub-optimality.
- *Improvement heuristics:* A heuristic method is used to obtain a solution which is improved via mathematical programming models.
- *Branch-and-price/column generation-based approaches:* In this approach, the aim is to achieve convergence faster by modifying the exact method used.
- *Relaxation-based approaches:* The concept of this approach is to identify attributes of an exact method that significantly slow down the convergence and relax them.

From the above classifications, we will focus and elaborate on decomposition approaches and provide application examples. Decomposition approaches are specifically useful for complex and integrated problems. Dividing the master problem into sub-problems that are handled and solved independently reduces the complexity of the problem and makes it easier to obtain a feasible solution for the master problem. In the context of matheuristics, mathematical models are used to solve some or all of the sub-problems. In cases of large scale real-world problems a time limit can be applied to prematurely stop the exact method and, thus, avoid excessive computational times. A *two-phase approach* is a subclass of decomposition approaches in which the matheuristic algorithm divides the master problem into two phases that are solved separately [39]. This approach is in line with collaborative combinations as described in [38]. A few examples of studies that have implemented a two-phase decomposition matheuristic approach are described next. [40] studied a pickup and delivery problem involving daily routes assignment of logging trucks in forestry which was solved in two phases. In the first phase, an exact model was used to obtain the optimal flow from supply points to demand points, and transport nodes were created based on the obtained solution. Next, in phase two, they used a heuristic method to combine the transport nodes into actual routes. In the study of [41], a logistic problem arising in disaster response activities was decomposed into two parts that were solved separately. In the first part, the authors obtained location decisions and approximate vehicle routes by solving exactly a simplified version of the complete mathematical formulation. Then, in the second part, they implemented an algorithm that generated pick up and delivery schedule for each vehicle based on the information obtained from the previous phase, and built detailed vehicle routes by solving a system of linear equations. Lastly, [42] studied the school bus routing problem in which the set of visiting stops needs to be determined, the stop where each student has to walk to needs to be defined, and the school bus travelling distance needs to be minimised. The problem was solved in two phases: in phase one, a heuristic method was used to solve the routing part of the problem, and in phase two, an exact model is used to assign students to stops. Further examples of matheuristic applications including additional classifications can be found in the survey of [43].

REFERENCES

[1] H. P. Williams, "Integer programming," in *Logic and Integer Programming*. Springer, 2009, pp. 25–70.

[2] R. J. Vanderbei, "Linear programming: foundations and extensions," *Journal of the Operational Research Society*, vol. 49, no. 1, pp. 94–94, 1998.

[3] M. Conforti, G. Cornuéjols, G. Zambelli, M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer programming models*. Springer, 2014.

[4] C. H. Papadimitriou, "On the complexity of integer programming," *Journal of the ACM (JACM)*, vol. 28, no. 4, pp. 765–768, 1981.

[5] M. R. Garey, D. S. Johnson *et al.*, "A guide to the theory of np-completeness," *Computers and intractability*, pp. 37–79, 1990.

[6] C. A. Floudas and X. Lin, "Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications," *Annals of Operations Research*, vol. 139, pp. 131–162, 2005.

[7] D. M. Ryan and B. A. Foster, "An integer programming approach to scheduling," *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pp. 269–280, 1981.

[8] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.

[9] A. Montero, I. Méndez-Díaz, and J. J. Miranda-Bront, "An integer programming approach for the time-dependent traveling salesman problem with time windows," *Computers & Operations Research*, vol. 88, pp. 280–289, 2017.

[10] A. Billionnet and É. Soutif, "Using a mixed integer programming tool for solving the 0–1 quadratic knapsack problem," *INFORMS Journal on Computing*, vol. 16, no. 2, pp. 188–197, 2004.

[11] A. Lodi and M. Monaci, "Integer linear programming models for 2-staged two-dimensional knapsack problems," *Mathematical Programming*, vol. 94, no. 2, pp. 257–278, 2003.

[12] I. Litvinchev, L. Infante, and L. Ozuna, "Approximate packing: integer programming models, valid inequalities and nesting," *Optimized Packings with Applications*, pp. 187–205, 2015.

[13] K. Park, S. Kang, and S. Park, "An integer programming approach to the bandwidth packing problem," *Management science*, vol. 42, no. 9, pp. 1277–1291, 1996.

[14] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," *Design Automation for Embedded Systems*, vol. 2, pp. 165–193, 1997.

[15] C. Canel and B. M. Khumawala, "A mixed-integer programming approach for the international facilities location problem," *International Journal of Operations & Production Management*, vol. 16, no. 4, pp. 49–68, 1996.

[16] J. Kratica, D. Dugošija, and A. Savić, "A new mixed integer linear programming model for the multi level uncapacitated facility location problem," *Applied Mathematical Modelling*, vol. 38, no. 7-8, pp. 2118–2129, 2014.

[17] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao, "An integer programming framework for critical elements detection in graphs," *Journal of Combinatorial Optimization*, vol. 28, pp. 233–273, 2014.

[18] H. Meier, N. Christofides, and G. Salkin, "Capital budgeting under uncertainty—an integrated approach using contingent claims analysis and integer programming," *Operations Research*, vol. 49, no. 2, pp. 196–206, 2001.

[19] R. Mansini, W. Ogryczak, M. G. Speranza, and E. T. A. of European Operational Research Societies, *Linear and mixed integer programming for portfolio optimization*. Springer, 2015, vol. 21.

[20] C. Lemaréchal, "Lagrangian relaxation," *Computational combinatorial optimization: optimal or provably near-optimal solutions*, pp. 112–156, 2001.

[21] J. Kendall, "Hard and soft constraints in linear programming," *Omega*, vol. 3, no. 6, pp. 709–715, 1975.

[22] T. A. Jessin, S. Madankumar, and C. Rajendran, "Permutation flowshop scheduling to obtain the optimal solution/a lower bound with the makespan objective," *Sādhanā*, vol. 45, no. 1, p. 228, 2020.

[23] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?" *Journal of Computer and System Sciences*, vol. 37, no. 1, pp. 79 – 100, 1988.

[24] P. Hansen and N. Mladenović, "Variable neighborhood search: Principles and applications," *European Journal of Operational Research*, vol. 130, no. 3, pp. 449 – 467, 2001.

[25] M. Pirlot, "General local search methods," *European Journal of Operational Research*, vol. 92, no. 3, pp. 493 – 511, 1996.

[26] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*. Kluwer Academic Publishers, 2002, pp. 321–353.

[27] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[28] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, vol. 285, no. 2, pp. 405–428, 2020.

[29] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intell. Data Anal.*, vol. 12, no. 1, pp. 3–23, Jan. 2008.

[30] A. Almutairi, E. Özcan, A. Kheiri, and W. G. Jackson, "Performance of selection hyper-heuristics on the extended hyflex domains," in *Computer and Information Sciences*, T. Czachórski, E. Gelenbe, K. Grochla, and R. Lent, Eds. Cham: Springer International Publishing, 2016, pp. 154–162.

[31] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, Dec 2013.

[32] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," in *Adaptive and multilevel metaheuristics*. Springer, 2008, pp. 3–29.

[33] G. Kendall and M. Mohamad, "Channel assignment optimisation using a hyper-heuristic," in *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, vol. 2, Dec 2004, pp. 791–796.

[34] J. H. Drake, E. Özcan, and E. K. Burke, "An improved choice function heuristic selection for cross domain heuristic search," in *Parallel Problem Solving from Nature - PPSN XII*, C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 307–316.

[35] L. N. Ahmed, E. Özcan, and A. Kheiri, "Solving high school timetabling problems worldwide using selection hyper-heuristics," *Expert Systems with Applications*, vol. 42, no. 13, pp. 5463 – 5471, 2015.

[36] G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *Journal of Computational Physics*, vol. 104, no. 1, pp. 86 – 92, 1993.

[37] E. K. Burke and Y. Bykov, "A late acceptance strategy in hill-climbing for examination timetabling problems," in *In Proceedings of the conference on the Practice and Theory of Automated Timetabling(PATAT*, 2008.

[38] J. Puchinger and G. R. Raidl, "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification," in *First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005*, J. R. Á. José Mira, Ed., vol. 3562. Las Palmas, Spain: Springer-Verlag, Jun. 2005, pp. 41–53.

[39] C. Archetti and M. Speranza, "A survey on matheuristics for routing problems," *EURO Journal on Computational Optimization*, vol. 2, no. 4, pp. 223–246, 2014.

[40] P. Flisberg, B. Lidén, and M. Rönnqvist, "A hybrid method based on linear programming and tabu search for routing of logging trucks," *Computers & Operations Research*, vol. 36, no. 4, pp. 1122–1144, 2009.

[41] W. Yi and L. Özdamar, "A dynamic logistics coordination model for evacuation and support in disaster response activities," *European journal of operational research*, vol. 179, no. 3, pp. 1177–1193, 2007.

[42] P. Schittekat, J. Kinable, K. Sörensen, M. Sevaux, F. Spieksma, and J. Springael, "A metaheuristic for the school bus routing problem with bus stop selection," *European Journal of Operational Research*, vol. 229, no. 2, pp. 518–528, 2013.

[43] M. O. Ball, "Heuristics based on mathematical programming," *Surveys in Operations Research and Management Science*, vol. 16, no. 1, pp. 21–38, 2011.