Surveys in Operations Research and Management Science 16 (2011) 21–38

Contents lists available at ScienceDirect

Surveys in Operations Research and Management Science

journal homepage: www.elsevier.com/locate/sorms

Review

Heuristics based on mathematical programming

Michael O. Ball *

Robert H Smith School of Business and Institute for Systems Research, University of Maryland, College Park, MD 20742, United States

ARTICLE INFO

Article history:
Received 2 July 2010
Accepted 6 July 2010

ABSTRACT

This paper provides a survey of heuristics that make use of mathematical programming models and methods. The first class of methods covered break down a problem into a sequence of subproblems where each subproblem is modeled as a mathematical program and solved optimally. The second class of methods are improvement algorithms that solve a mathematical program to generate an improved solution from a known feasible solution. This class of techniques is also referred to as large-scale neighborhood search. The third class of methods considered employ a mathematical programming algorithm, most notably branch-and-bound, to generate an approximate solution to the problem of interest. Finally, we consider methods that solve a relaxation to the original problem of interest as a first step in generating a good feasible solution.

Contents

1. Introduction

Mathematical programming involves the study of techniques that can generate provably optimal solutions to optimization

* Tel.: +1 301 405 2227; fax: +1 301 405 8655.
E-mail address: mball@rhsmith.umd.edu.

1876-7354/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.sorms.2010.07.001

problems. From a practical standpoint, the field of heuristics has a very similar goal, i.e. to generate "solutions" to optimization problems. The difference is that the solutions should be "good" but not necessarily provably optimal. In many practical applications the distinction can be a minor one. Broadly speaking the goal of this paper is to explore the relationship between these two approaches to problem solving. Our emphasis will be on how mathematical programming (MP) can be used for practical, approximate problem solving, rather than on a theoretical study of the approximate use of mathematical programming techniques.

The first heuristics were based on the iterative application of very simple techniques that either sequentially constructed a solution or sequentially improved a feasible solution. The use of very simple component steps resulted in very fast algorithms. As the field of mathematical programming grew, fast (optimal) algorithms were developed for certain classes of problems. A natural extension of the original heuristic design philosophy was to consider embedding optimization algorithms as subprocedures within a more complex algorithm. In effect the heuristic designer's "bag of tricks" was expanded to include not only simple techniques, e.g. a two-for-two arc exchange, but also more complex heuristic methods, e.g. a shortest path algorithm.

Two general classes of heuristics are construction heuristics and improvement heuristics. Construction heuristics start from "scratch" and proceed through a set of steps, each of which adds a component to the solution until a complete (feasible) solution is generated. Section 2 covers construction heuristics, where one or more of the steps involves the solution of a mathematical programming problem. We also label such methods decomposition approaches since they effectively decompose a larger problem into a series of sequentially executed subproblems. Improvement heuristics start with a feasible solution and iteratively execute solution improving steps until some termination condition is met. Section 3 covers improvement heuristics where the solution improving step involves the solution of a mathematical programming problem.

An exact optimization algorithm terminates with an optimal solution and a proof of optimality. In many cases, a significant portion of the total solution time is spent proving that a solution found (quickly) is optimal. Another common scenario is that a large amount of computing time is spent going from a "near-optimal" solution to an optimal one. With this motivation, in many practical settings, exact mathematical programming algorithms are modified to generate a very good, but not necessarily optimal, solutions. This class of heuristic approaches in covered in Section 4.

It is very often the case that while a problem may be very difficult, a certain relaxation to that problem may be efficiently solvable. The solution to a relaxation generates a bound on the value of a problem's optimal solution. As such relaxations are often employed in exact mathematical programming approaches. Additionally, they can often serve as a basis for effective heuristics. Two general approaches are used. In one, the solution to a relaxation is modified to generate a feasible solution to the problem of interest. Probably the prototypical approach of this type involves rounding of the solution to a linear programming relaxation of an integer program. The second class of relaxation based approaches makes use of the dual information provided by the solution to the relaxation in a subsequently executing heuristic. Section 5 covers relaxation based approaches.

A large body of literature, e.g. [1,2], has been devoted to so-called primal–dual heuristics and their associated performance guarantees. One could possibly classify these approaches as MP-based heuristics, however, we will not review that literature here. Similarly, there is a broad class of MP-based heuristics that employ stochastic dynamic programming. We will not cover these methods here as they are covered in other recent books and surveys, e.g. [3]. As noted earlier, out emphasis is on practical problem solving and so we will not review the extensive literature on worst case performance guarantees, although occasionally we touch on such results to help us compare heuristics.

2. Decomposition approaches

In a certain sense, whenever a mathematical programming algorithm is applied within an application context, it becomes a heuristic since an exact algorithm exactly solves an abstract problem with a specific set of assumptions. Such assumptions will never be totally satisfied in practice. In particular, it is very often the case that inputs to an optimization model are the outputs of another decision process and the outputs of the optimization model are the inputs to yet another decision process. When viewed in a narrow context an optimal solution is found and used, but when viewed in the context of the broader decision environment the mathematical programming algorithm used to find the intermediate (optimal) solution is a component of a "larger" heuristic. For example, in the context of airline crew and fleet planning, the crew pairings problem might be solved optimally using integer programming. Yet, the input to the crew pairings problem depends on the aircraft fleet plan as well as the passenger schedule. The pairings themselves are just a component of the crew's monthly schedule. Thus, when viewed from the perspective of crew pairings problem the use integer programming represents an application of mathematical programming. However, in the context of fleet and crew planning integer programming is a component in an overall heuristic solution. This is an example of a mathematical programming based heuristic that employs a particular heuristic problem decomposition.

There are generally two sets of considerations in designing decomposition approaches. The first is, of course, what the structure of the decomposition should be. That is, what sequence of problem should be solved to finally arrive at a complete solution. In the case of the fleet and crew planning, the decomposition described above is a very natural one that fits in well with manual planning processes, data flows and databases, etc. Such a decomposition has certain practical advantages but might not lead to the "best" overall solutions. In other contexts, there may not be a natural decomposition so the heuristic designer is a liberty to choose one based purely on algorithm design considerations.

The second set of considerations has to do with specializing each optimization model so that the solution output in an "upstream" step leads to a high quality solution in "downstream" steps. Alternatively, or in addition, the process could also contain types of feedback loops where certain steps are re-executed based on an evaluation of solution quality.

2.1. Decomposition heuristics for the TSP and VRP

We start with a presentation of decomposition approaches for certain routing problems including the traveling salesman problem (TSP). The TSP is defined on a set of nodes, $1, 2, \ldots, n$, with a cost, c_{ij}, defined for each (unordered) pair of nodes, $\{i, j\}$. The problem is to find a *tour* that visits each node exactly once and has minimum total cost. In the context of vehicle routing, we can view this as the problem faced by a delivery vehicle that must visit a set of n customers. The problem is to sequence the customers so as to minimize overall cost, which could represent distance, time or some more complex routing function. For the more general vehicle routing problem (VRP) [4], there are m vehicles, each vehicle has a capacity b and each stop, i has a demand level d_i. One additional node, identified as node 0, is designated as the depot. The problem is to allocate customer nodes to each vehicle so that the total demand allocated to each vehicle is no more than b and to find a tour for each vehicle that covers the customer nodes allocated to it and that starts and ends at the deport. The objective function as before is to minimize the total costs of all vehicle routes. Clearly both of these problems can be viewed in terms of a undirected network where each node pair is connected by an undirected arc of length c_{ij}.

There is an obvious close relationship between the TSP and the VRP since the TSP can be viewed as a one-vehicle VRP and the vehicle tours in any optimal VRP solution should individually be optimal TSP tours. We start our discussion of decomposition heuristics for the VRP with the TSP and show how certain decomposition approaches for the TSP strongly motivate related decomposition approaches for the VRP.

The TSP optimization based decomposition heuristics make use the strategy of approaching a "difficult" optimization problem by solving a related "easy" optimization problem. The TSP can be viewed as the problem of connecting the nodes in a network using a minimum cost set of arcs that forms a tour. The problem of finding a minimum cost set of arcs that connects the nodes in a network but has no other properties is known as the minimum spanning tree (MST) problem. Intuitively one would expect the solutions to these problems to be related. In fact, there is a very strong relationship for the case where arc costs represent Euclidean distances or more generally satisfy the triangle inequality.

Consider Fig. 1, which illustrates the following heuristic [5]:
Tree heuristic

1. find an MST;
2. duplicate each arc in the MST;
3. repeat until all nodes have degree 2:
   find a node with degree greater than 2; reduce the node degree by 2 by replacing a two-arc path with a single arc path with the same end-nodes.

It is easy to see that a tour/TSP solution is obtained. If z^*_{MST} is the cost of an optimal MST solution, z^*_{TSP} is the cost of an optimal TSP solution and $z_{H(T)}$ is the cost the solution obtained by the tree heuristic just defined then it can easily be shown that, if costs satisfy the triangle inequality then,

$$z_{H(T)} \leq 2z^*_{MST} \leq 2z^*_{TSP}.$$

That is, the tree heuristic has a worst case bound of 2. In fact, we can further improve this bound by taking an optimization based approach to the second step of the above procedure. It is easy to see that the process of replacing paths of length 2 with arcs can be applied to any spanning Eulerian tour, i.e. a spanning connected subgraph in which all nodes have even degree. The *Matching-Tree Heuristic* due to Christofides [6], replaces step 2 in the tree heuristic with a step that finds a minimum cost set of arcs, which when added to a tree, forms a spanning Eulerian tour. This problem can be formulated as a matching problem. The Matching-Tree Heuristic is obtained from the Tree Heuristic by replacing step 2 with:

$2'$ solve a minimum cost matching problem over the network induced by the nodes of odd degree in the MST; add the arcs in the matching solution to the MST;

If we define $z_{H(MT)}$ as the cost of the solution obtained by the Matching-Tree Heuristic then it can be shown that:

$$z_{H(MT)} \leq 3/2z^*_{TSP}.$$

These TSP heuristics and the related analysis motivated a class of MP-based heuristics for the VRP. Consider the following representation of the VRP:

$$\text{Min: } \sum_j \hat{c}_{TSP}(S_j) \tag{1}$$

$$\text{s.t. } \sum_j x_{ij} = 1 \quad \text{for all customers } i \tag{2}$$

$$\text{s.t. } \sum_i x_{ij} \leq b \quad \text{for all vehicles } j \tag{3}$$

$$S_j = \{i : x_{ij} = 1\} \quad \text{for all vehicles } j \tag{4}$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } i \text{ and } j. \tag{5}$$

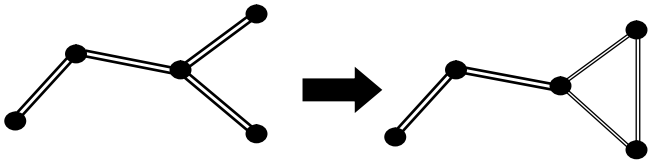

Fig. 1. Tree Heuristic: replacing a two-arc path with a single arc path.

Here $\hat{c}_{TSP}(S)$ is the cost of a (optimal) TSP tour over node set $S \cup \{0\}$. From this formulation we can see that constraints (2), (3) and (5) define the so-called *generalized assignment problem* (GAP). Thus, it is appealing to solve an instance of the GAP and then to solve several TSPs to "evaluate" the objective function. This general heuristic strategy predates the development of mathematical programming based heuristics and is known as the "cluster-first, route-second" strategy (see for example, [7]). The challenge of such an approach is that there is no obvious objective function to use for the GAP. The "trick" is to find a surrogate cost function that gives a good approximation for the TSP costs. The preceding TSP analysis provides insights into approximation approaches.

Fisher and Jaikumar [8] were the first to introduce an MP-based approach of this type. They executed an initial step in which a "seed node" s_j was chosen for each route j. The cost of assigning a customer i to route j, d_{ij}, was defined as c_{is_j}. The overall approach can be defined as:
GA heuristic

1. Choose seed nodes, s_j for $j = 1, \ldots, m$;
2. Solve the GA: Min $\{\sum_{ij} d_{ij}x_{ij}$: (2), (3), (5)$\}$
3. Solve a TSP over each cluster $S_j \cup \{0\}$ defined in step 2.

If one duplicates each arc in the tree associated with a seed node and the customers assigned to it, then the same "shortcutting" strategy used in steps (2) of the Tree Heuristic or step $(2')$ of the Matching-Tree Heuristic could be used to convert each seed node tree into a tour for the corresponding vehicle. Using a similar argument, the cost of the tour would be no more 2 times or resp. 3/2 times the cost of the seed node tree. This provides evidence that the cost function used for the GA in Step 2 is a good approximation. Of course, a deficiency of this approach is the potentially heavy dependence on the seed selection step. This weakness was alleviated in the method proposed by Bramel and Simchi-Levi [9,10]. They proposed combining steps (1) and (2) so that seed nodes are chosen and clustering carried out as part of the same optimization problem. The underlying optimization problem is known as the concentrator location problem so this approach is call the Location Heuristic.

Location Heuristic

1. Choose a set of *candidate* seed nodes;
2. Solve a concentrator location problem to determine a seed node, s_j and a cluster, S_j for each vehicle j.
3. Solve a TSP over each cluster $S_j \cup \{0\}$ defined in step 2.

Bramel and Simchi-Levi analyze a variant of the problem in which the number of vehicles used is a variable, which naturally grows as the number of customers grows. They show that the Location Heuristic is asymptotically optimal for this problem. This analysis depends on the property that the surrogate cost function used in the Location Heuristic to evaluate a cluster is bounded by a constant times the associated optimal tour length.

One could argue that certainly the Tree Heuristic and the Matching-Tree Heuristic have substantial appeal since they decompose an NP-hard optimization problem (the TSP) into a sequence of polynomially solvable optimization problems. On the other hand the GA Heuristic and the Location Heuristic decompose an NP-hard optimization problem (the VRP) into a sequence of (different) NP-hard optimization problems. Is this progress? In fact

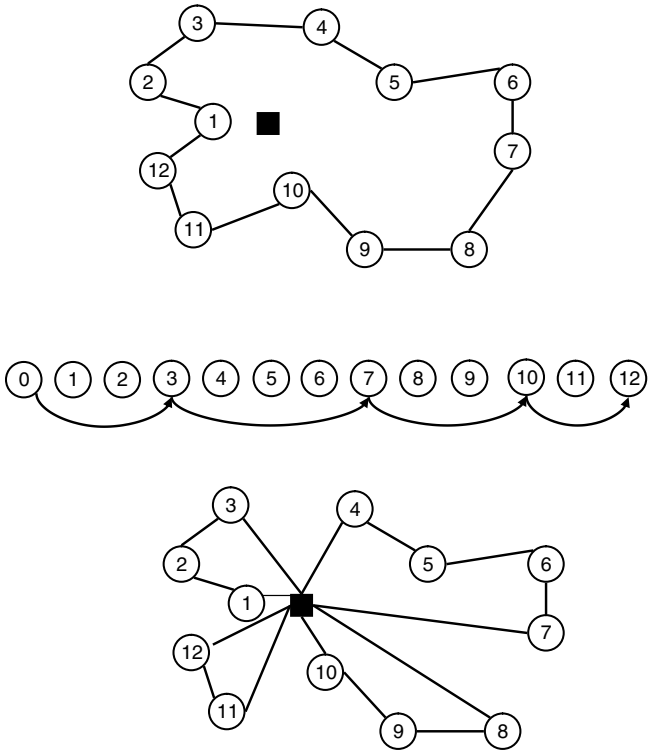

Fig. 2. Route-First, Cluster-Second Heuristic.

from a practical standpoint these heuristics have performed well. It is well known that not all NP-hard problems are equally hard in practice. In fact large instances of the GA, the concentrator location problem and the TSP have been solved to optimality whereas this is not the case for the VRP.

One might ask whether other decomposition approaches to the VRP could be equally successful. Some early VRP heuristics took an approach complimentary to cluster-first, route-second, namely, route-first, cluster-second, as defined below (see Christofides [7]):

Route-First, Cluster-Second Heuristic

1. Find a TSP tour through the entire set of customer nodes;
2. Let the TSP customer sequence be defined by $\{i_1, i_2, \ldots, i_n\}$. Choose a set of m "cut points": $k_1 < k_2 < \cdots < k_m$. Let the VRP vehicle clusters be defined by: $S_1 = \{i_{k_1}, \ldots, i_{k_2-1}\}$, $\ldots, S_{m-1} = \{i_{k_{m-1}}, \ldots, i_{k_m-1}\}, S_m = \{i_{k_m}, \ldots, i_n, i_1, \ldots, i_{k_1-1}\}$. Form the vehicle sequences by inserting the depot at the beginning of each cluster.

As is illustrated in Fig. 2, Step 2 of the Route-First, Cluster-Second Heuristic can be solved optimally using a shortest path model. Suppose that the nodes are labelled 1 through n in the order in which they appear in the TSP tour, where the start node 1 is chosen arbitrarily. Then, a shortest path network is constructed on node set 0 through n where are an arc from node i to node j, with $i < j$, represents the VRP route starting at the depot, proceeding to node $i + 1$ then following the TSP node order to node j and returning to the depot. The cost placed on each arc is the cost of the corresponding VRP route. The shortest path from node 0 to node n gives the minimum cost VRP solution that is (1) consistent with the TSP ordering and (2) has a route starting with node 1. By varying the start node the best partition over all start nodes can be obtained. We note that this tour partitioning problem is an instance of a set partitioning problem with consecutive ones with wrap-around — see [11,12].

While this decomposition approach would seem to provide an appealing alternative to the previous heuristics, in fact, it has generally not performed as well from a computational perspective and has only been used in practice in specialized settings. We offer some insight into why this might be the case. First from the point of view of computational efficiency, the TSP solved in step 1 of the Route-First, Cluster-Second Heuristic involves the entire customer set and as such it can be quite large and difficult to solve. This is to be contrasted with the much smaller TSP solved in the case of the Cluster-First, Route-Second Heuristics. Second, While solving a TSP in the first step captures routing costs in a certain sense, the costs of traveling to and from the depot are not well captured. In fact, this property has led to an interesting asymptotic result (see [13] and the related Ref. [14]), which states that there is no asymptotically optimal Route-First, Cluster-Second Heuristic for the VRP.

2.2. Examples and strategies

Transit crew and vehicle scheduling and airline fleet and crew planning represent two application areas for which there has been substantial fundamental research into exact and approximate solution strategies as well as substantial implementation success. See references [15–17] for further discussion of aviation applications and references [18–21] for further discussion of transit applications. Furthermore, these problem areas are quite similar but at the same time have subtle differences that make their study and comparison enlightening. Both have three basic schedules that must be produced, namely, a passenger schedule, a vehicle schedule and a crew schedule. In addition, the standard practice used in both cases is a corresponding problem decomposition, namely: (1) create passenger schedule; (2) create vehicle schedule; (3) create crew schedule. Algorithm and system designers have employed a variety of approaches within this general structure, which vary in terms of the techniques used to solve each step and also in the manner in which the solutions to each step are coordinated. It is also instructive to note that even though we are focusing on the combined solution of three very complex problems, even these cannot be considered in isolation in that the passenger schedule depends, in the airline case, on the choice of cities to which the airline provides service, the locations of maintenance facilities, access to airport time slots and gates, etc. In the transit case, there is a problem of line layout that precedes scheduling. In both cases, there are related problems involving investment in equipment, hiring of crews, etc.

Passenger schedule creation

The passenger schedule defines the service provided to customers. It is typically represented in terms of a timetable indicating when service is provided between service locations. In the airline case the key consideration is the capture of revenues and in the transit case, the key consideration is the provision of a certain level of service. Of the three problem areas, this one has proved to be the least amenable to attack by formal solution methods. The reasons for this include problem size and complexity and difficulties in collecting data and quantifying the quality of solutions. A typical transit schedule consists of basic line service in which trips are periodically made from one end of the line to the other. There usually is a high degree of structure, e.g. a trip is made every 15 min. Simple decision support tools as well as graphical aids are usually provided to aid the design process. The problem becomes more complex when additional "express" trips are added during peak demand periods. Also, complexity arises when increasing or decreasing service frequency over the course of a day. Although timetable creation is ostensibly a separate step, independent of vehicle scheduling, vehicle considerations are clearly taken into account. For example, the basic service creation process would typically start a trip from the end of a line shortly after a trip terminating at that end point was completed. Thus, there is an implicit vehicle schedule created and, in fact,

decision support tools typically report on the number of vehicles required for various proposed timetables. On the other hand, the direct connection between passenger schedule creation and vehicle scheduling breaks down as express trips are added and service frequency is changed. Although in the airline case, there is less structure to the service provided, the level of formal model support as well as the implicit connection to vehicle scheduling is similar (see [15]).

Vehicle scheduling

Vehicle scheduling for transit and aviation possess the same basic inputs. That is, for transit, the trips and, for aviation, the flight legs are demand entities that can be represented by start and end locations and start and end times. These can be interconnected within a flow network where an arc from one such entity to another indicates that the same vehicle can service both. The simplest version of this problem, which was originally defined and analyzed in an early paper of Dantzig and Fulkerson [22] (see also [11,23]), is a minimum cost flow problem. In the aviation case, e.g. [16], it becomes a multi-commodity flow problem, due the presence of multiple aircraft types and, in the transit case, it also become more complex due to the presence of multiple vehicle garages. In the aviation case, vehicle scheduling requires a second step, the so-called aircraft maintenance routing problem, which determines multi-day schedules that include required visits to a maintenance base. Some efforts to incorporate crew considerations within vehicle scheduling include the inclusion of long layovers in transit vehicle schedules to accommodate certain types of crew rest periods. In a somewhat similar vein, aircraft schedule creation has been adjusted to encourage desired crew "short connects" that can be used by assigning the corresponding flight legs to the same aircraft [15]. It is interesting to note that both in the transit and aviation cases [24], vehicle scheduling models have been formulated that allow certain adjustments to the timetables. Specifically, inter-trip/flight leg arcs can be defined that correspond to adjusting the start time of a trip. For example, if a trip with a 4:00 PM start time could not be preceded by an arc whose associated aircraft was not available until 4:07. The inclusion of such an arc and its subsequent selection would imply that the start time of the trip had to be moved until 4:07. This technique can only be used with relative small time perturbations; otherwise the underlying network structure will break down.

Crew scheduling

In both the transit and aviation contexts, crew scheduling is broken down into a two-step process. In transit, the first step produces daily schedules, while in aviation, the first step produces pairings, which are crew work periods that start and end at a crew base. In both cases, the second step creates rosters or bidlines, which specify a work plan over a longer period, usually one month in length. Set partitioning and covering approaches are used extensively in both contexts. We note that earlier approaches [25,26] to the daily transit scheduling problem, used a decomposition, which involved a first step that breaks vehicle schedules into 1/2 day pieces, approximately 4 h in length, and a second step, which paired pieces into a full day schedules. The first step involved the decomposition of a long sequence of tasks (the vehicle schedule) into smaller tasks (the pieces). As such it possesses the sequential decomposition structure described in the previous section so it could be solved as a shortest path problem (see Fig. 3).

The second step could very naturally be formulated as a matching problem; however, the presence of certain special union constraints sometimes required the inclusion of side constraints [27]. Although shortest paths provide an efficient solution approach to the first step there is no natural objective function for this problem. The objective of the first step is simply to produce pieces that lead to a good solution to the second step. The approach described in [28,29], which is now part of the broadly implemented Hastus System, involves solving an approximate LP relaxation to the daily crew scheduling problem and then iteratively solving a sequence of shortest path problems that minimize the deviation between the values of the LP piece variables and the pieces generated. In [27], a feedback loop is used, where Lagrange penalties are generated and input into the shortest path problems, based on the structure of the complete crew schedules generated in the previous iteration.

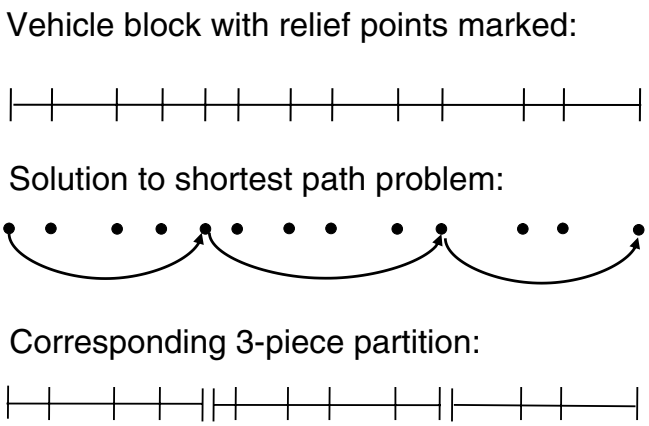

Fig. 3. Partitioning vehicle blocks using shortest paths.

Integrated approaches

Although the "vehicles first, crews second" is the most natural problem decomposition, research has been directed at the alternate decomposition of "crews first, vehicles second". Such models, e.g. [30–33] for aviation, first solve a crew scheduling step that specifies connections between certain trips or flight legs based on a crew-cost objective function and then in a second step generates a complete vehicle schedule. We note that such approaches are particularly appealing in the transit case, where crew costs dominate vehicle costs [12]. Freling et al. [30] compare three approaches to transit crew and vehicle scheduling: (1) the standard sequential decomposition described above; (2) a sequential approach that determines crew schedules first and vehicle schedules second; and (3) an approach that simultaneously determines crew and vehicle schedules. All approaches employ a Lagrangian relaxation based set partitioning model. Two cases are analyzed. For the less constrained "changeover" case, models (2) and (3) provide little advantage over model (1). However, for the "no changeover" case the improvement is more dramatic. This seems to validate the folklore that the traditional sequential approach is an effective heuristic decomposition, except in situations where severe or unusual crew scheduling constraints are imposed. Most recently there have been efforts to combine and coordinate various steps. Other research has focused on integrating or partially integrating various of the other steps mentioned, e.g. [34–37].

2.3. Perspectives on decomposition strategies

The research outlined above as well as other work provides certain general concepts that can applied in a variety of settings.

Choice of decomposition and subproblems to solve

Efficiency: The fundamental reason for decomposing a problem is that the larger problem is too difficult to solve directly. Thus, it is essential that the individual problems in the decomposition each be efficiently solvable. This should either mean that there exists a polynomial solution algorithm but it can also mean that the underlying (NP-hard) problem is small or amenable to rapid solution for some other reason.

Effective approximation: The sequence of problems solved should in the end yield a high quality overall solution. For this to occur the "upstream" problems should either explicitly or implicitly capture the major downstream costs. For example, the Generalized Assignment heuristic includes a surrogate objective function that approximates second stage routing costs. In the case of transit scheduling, one reason that the "vehicles first, crews second" strategy has continued to be used extensively is that the objective of minimizing vehicle time implicitly also minimizes the time crews are required to be on duty.

Consistency with operational processes: While the algorithm designer should, in theory, create the decomposition that leads to the "best" solutions, it can be the case that the underlying operational processes impose constraints and/or provide advantages to using "natural" decompositions. For example, it is certainly the case that the passenger schedule – vehicle schedule – crew schedule decomposition for both transit and aviation is ingrained in the underlying business operations. For example, within an airline, it is typically the case that passenger schedule generation is the purview of the marketing department since this directly relates to revenues while the other problems reside in operations departments. While such natural decompositions can lead to significant business inefficiencies and in such cases, alternatives should be proposed, if operational processes allow manual adjustments to the results of one process step then the automatic processes should be able to respond to such adjustments. Furthermore, it can be the case, that, at least in the short term, such decompositions are imposed as hard organizational constraints.

Model integration

We now list some general strategies for coordinating the interdependent decisions addressed by a sequence of models.

Surrogate objectives and constraints: We have seen that a variety of techniques are used to capture the downstream impact within an upstream procedure. This usually takes the form of formulating surrogate cost functions that represent desirable downstream problem effects. Alternatively, it is sometimes possible to impose constraints in an upstream model that eliminate the occurrence of undesirable or infeasible downstream solution characteristics. Much of the recent research aimed at creating integrated models still results in problem decomposition with formal decomposition schemes such as Benders decomposition and Lagrangian relaxation providing coordination mechanisms (e.g. see [36]).

Incorporating upstream decision variables in downstream model: When solving a problem that takes as input the output of an upstream model, it is sometimes possible to include variables in the model that represent slight perturbations of the upstream solution. A classic example of this approach is the inclusion of trip or flight leg windows within vehicle scheduling models.

Feedback: Certain steps in a basic decomposition can sometimes be re-executed in a feedback loop to generate improved solutions. Feedback information can be provided using Lagrange penalties or dual information, e.g. [27]. Another simple approach alternatively solves an upstream and downstream model where each is constrained by the solution to the other. To illustrate this process, suppose a model has variable set (x, y) with x set by an upstream model, M_1 and y set by a downstream model, M_2. Then, M_1 would initially assign $x = x_0$, M_2 would then generate a solution (x_0, y_0) where $x = x_0$ is taken as a constraint. The process then iterates between M_1 and M_2, generating a sequence of solutions: (x_1, y_0), (x_1, y_1), (x_2, y_1), etc. See [38] for examples of this approach.

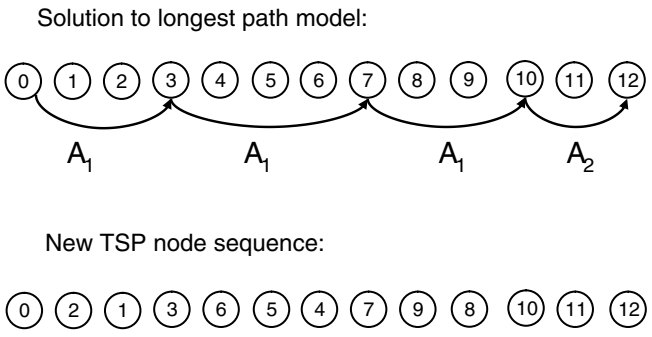

Fig. 4. Longest path model to find the best set of node exchanges.

3. Improvement heuristics

There is a vast literature on improvement heuristics. Improvement heuristics are particularly appealing since they can be combined with any other method that finds a feasible solution.

3.1. Large-scale neighborhood search

Neighborhood search represents a very effective and widely studied heuristic paradigm. The typical neighborhood search algorithm executes a relatively simple procedure associated with a relatively small neighborhood. On the other hand, optimization can be applied in this setting to search over much larger neighborhoods. Ahuja et al., [39] give an survey of these techniques. We describe two classes: in the first the neighborhood search uses shortest paths or dynamic programming and in the second the neighborhood search requires the identification of a negative cycle, which, depending on the application, is found using a variety of techniques including network flows and integer programming. We should note that it is perhaps possible to classify any improvement algorithm as neighborhood search. We have not done this and should note the some of the classes of techniques described by Ahuja et al. are described in other (non-neighborhood search) sections of this paper.

The solution to many problems can be characterized by a sequence items. In such cases, it is often possible to independently modify portions of an existing sequence. In such cases, a dynamic programming or shortest/longest path problem can be set up to choose a set of independent subsequences, where each subsequence is modified. The solution produced is the result of the set of modifications that provides the greatest solution improvement. We illustrate these ideas on the TSP. Suppose that the current TSP tour visits the set of nodes in numerical order: $1, 2, \ldots, n, 1$. We can view modifications to this tour as exchanges in this sequence. For example, exchanging nodes 2 and 5 would lead to the solution: $1, 5, 3, 4, 2, 6, \ldots, n, 1$. The cost savings associated with this exchange would be: $c_{12} + c_{23} + c_{45} + c_{56} - c_{15} - c_{53} - c_{42} - c_{26}$. Of course, it is clear that the portion of the tour involving nodes 6 through n is unaffected by this exchange and, in fact, an exchange among nodes in this portion could be evaluated and carried out independently the exchange between nodes 2 and 5.

As illustrated in Fig. 4, we can define a longest path problem on an acyclic network to find the best set of such exchanges as follows. The node set is $V = \{0, 1, \ldots, n\}$. There are two types of arcs. An arc $(i, j) \in A_1$ represents an exchange between nodes $i+1$ and $j-1$. An arc $(i, j) \in A_2$ represents that no exchanges are made along the sequence between i and j. Thus, $A_1 = \{(i, j) : 0 \le i \le j - 3$ with $j \le n\}$ and $A_2 = \{(i, j) : 0 \le i < j \le i + 2$, with $j \le n\}$. The weight of an arc $(i, j) \in A1$ is given by:

$$d_{ij} = c_{i,i+1} + c_{i+1,i+2} + c_{j-2,j-1} + c_{j-1,j} - c_{i,j-1} - c_{j-1,i+2} - c_{j-2,i+1} - c_{i+1,j}$$

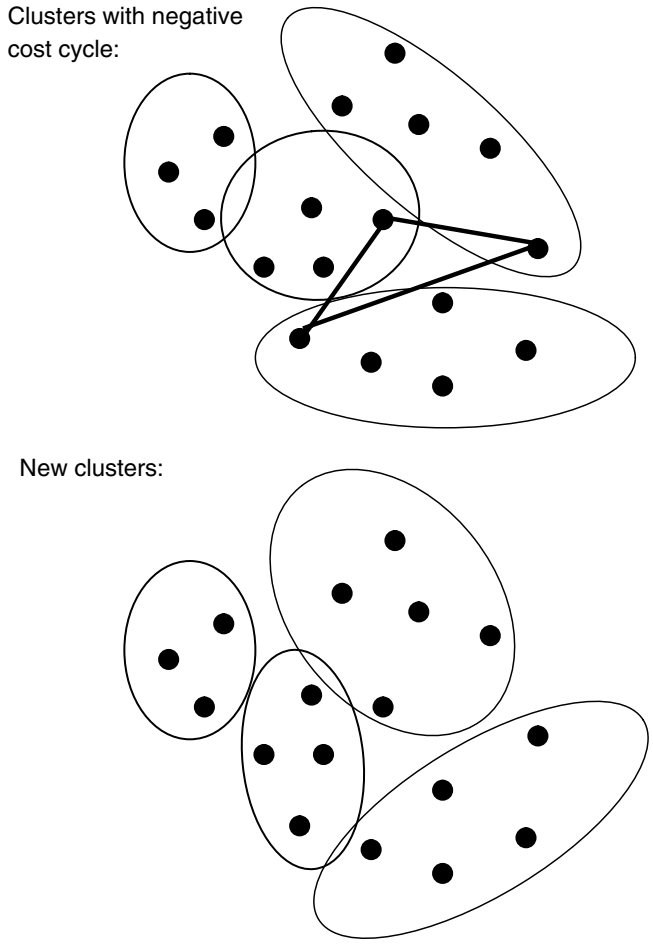

Fig. 5. Negative cycle exchange among clusters.

where c_{0k} is defined to be c_{nk} for all nodes k and, for the case of $i = j - 3$, $c_{j-2,i+1}$ and $c_{j-1,i+2}$ are replaced by $c_{j-1,i+1}$. The weight of an arc $(i, j) \in A_2$ is zero. A maximum weight path from node 0 to node n in the graph, $(V, A_1 \cup A_2)$ gives a set of independent exchanges that maximizes the total cost savings achievable over all such exchange sequences. Note that in general there will be many pairs of parallel arcs — the member of each pair of lower weight can be deleted. Also, negative weight arcs can be deleted. Although this is a longest path problem, which in general is NP-hard, it can be efficiently solved since the underlying network is acyclic. The approach outlined is a slight modification of the approach described in [39]. It was originally proposed by Potts and van de Velde [40], who have also applied it to machine scheduling problems. Other examples of the use of shortest paths, integer programming and dynamic programming to search neighborhoods can be found in [41,42].

A second general class of optimization based large-scale neighborhood improvement methods involves the identification of an improving cycle of exchanges. These are best illustrated on set partitioning/clustering problems. Fig. 5 illustrates an "exchange cycle", which represents a sequence of exchanges among multiple clusters. Whereas a typical simple exchange would transfer a node i from cluster A to cluster B in exchange for the transfer of a node j to cluster B, a 3-cycle exchange would transfer an $i \in A$ to B, a $j \in B$ to C and a $k \in C$ to A. Depending on the application, the evaluation of an exchange or sequence of exchanges might require consideration of both cost and constraints. For example in the VRP an exchange should improve the overall solution cost, but also, should not violate any vehicle capacity constraints. The simplest variant of the exchange sequence identification problem requires finding a "subset-disjoint" negative cycle. This problem has been shown to be NP-hard (see [43]). Nonetheless, this general approach has been applied in many contexts where the underlying cycle identification problem has special structure, is solved heuristically or is small enough to be solved using general purpose integer programming techniques (for examples see: [44–49]). Glover and his co-authors were early developers of heuristics based on this idea, which they call *the ejection chain approach* (see for example, [50,51]).

3.2. Finding the best solution over a restricted feasible region

For nearly all classes of optimization problems there is a steady stream of research on exact methods that continuously pushes the boundary on the size of problems that can be solved optimally. It would be very appealing if any such enhancements could be immediately put to use in heuristics that could be used to attack problems of arbitrary size. We now outline two general approaches that provide this capability. Using a mathematical programming point of view, we call the first *Row Partitioning* and the second *Restricted Column Set*.

Row partitioning
Input: feasible solution

1. cut out a portion of the solution of manageable size, i.e. a row subset;
2. apply an exact method over the portion cut out in (1);
3. paste the solution obtained in (2) back into the original solution;
4. if new, improved, feasible solution is created then repeat.

Restricted Column Set
Input: feasible solution

1. create a column/variable set by augmenting the columns/variables represented by the feasible solution with an additional set so that a efficiently solvable problem instance results;
2. apply an exact method over the column set created in (1);
3. if an improved solution is created then repeat.

These descriptions are, of course, just outlines of approaches. Their effectiveness will depend on the specifics of the problem addressed and the details of the implementation. We now discuss their application to specific problem settings.

3.2.1. Row partitioning

The most natural setting for applying Row Partitioning involves set partitioning problems or problems that can be conceptualized as a set partitioning problem (SP):

$$\text{SP: Min } cx \tag{6}$$
$$\text{s.t. } Ax = b \tag{7}$$
$$x \in \{0, 1\}. \tag{8}$$

Here, c is an arbitrary cost vector, A is an n x $m0/1$ matrix and b is an m-vector of 1's. Let $I = \{1, 2, \ldots, m\}$ and (I_1, I_2) be a partition of I so that $I_1 \cup I_2 = I$ and $I_1 \cap I_2 = \phi$. Let $b[1]$ and $b[2]$ be the vector b restricted to I_1 and I_2 respectively. Let J_i for $k = 1, 2$ be defined as $J_k = \{j : a_{ij} = 0 \text{ for } i \in I_{k'} \text{ where } k' = 2 \text{ if } k = 1 \text{ and } k' = 1 \text{ if } k = 2\}$, i.e. J_1 are those columns that are zero for $i \in I_2$ and J_2 is defined analogously. Let $A[1]$ and $A[2]$ be A restricted to J_1 and J_2 respectively and $x[1]$, $x[2]$, $c[1]$ and $c[2]$ be defined similarly. Then we may obtain a feasible solution to SP by solving SP[k] for $k = 1, 2$:

$$\text{SP[k] : Min } c[k]x[k] \tag{9}$$
$$\text{s.t. } A[k]x[k] = b \tag{10}$$
$$x[k] \in \{0, 1\}. \tag{11}$$

Of course, the question that immediately comes to mind is how to find such a partition. Probably the most common approach is

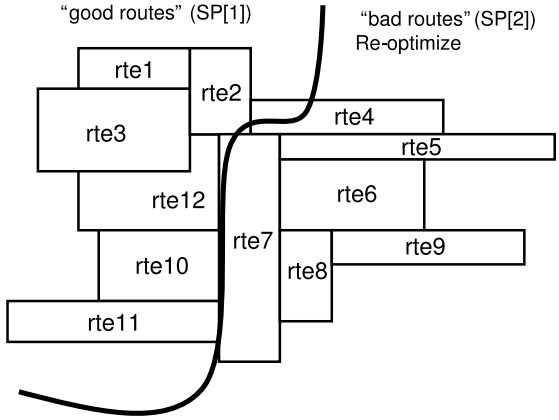

Fig. 6. Optimizing a "Bad Route" subset.

to base such a partition on an existing feasible solution in which case an improvement heuristic results. Starting with a feasible solution $\hat{x}$, one partitions the non-zero columns of $\hat{x}$ into two sets and defines I_k as those rows covered by each of the two members of the partitions. Each of SP[1] and SP[2] can be defined and the appropriate restricted math programs can be solved. This approach insures a feasible solution will result as the partition of the original solution provides a feasible solution to each of the subproblems. A further commonly used refinement of this approach is to partition the original feasible solution into a good portion and bad portion. Suppose that the bad portion leads to SP[2]. Then SP[2] is solved to provide a replacement for the bad portion. SP[1] is never solved as the good portion of the original solution is judged to be "good enough". In many cases, the partition is carried out with computational efficiency in mind. For example, the partition is defined so that the subproblem that is solved, SP[2] is small enough to be solved within a reasonable amount of computing time (It could easily be the case that SP[1] is too large to be efficiently solved).

We can thus define an improvement heuristic as follows:

R_ IMPROVE

1. For the current feasible solution $\hat{x}$, let $\hat{J}$ be the corresponding set of chosen columns.
2. Choose a subset J' of $\hat{J}$.
3. Solve SP[2], where I_2 are the rows covered by J'; let J^* by the chosen set of columns.
4. Replace J' with J^* to obtain a new $\hat{x}$; if stopping criterion met, then exit; otherwise go to 1.

This approach was used very early on to solve general set partitioning problems (see [52]). Over the years it has been applied in a number of specific contexts. The general area of vehicle routing has provided several domains for the successful application of this approach. Shaw [53] was one of the first to apply this general technique to VRP's. In [54], this approach is applied to an arc routing problem, which is a vehicle routing problem where demand is associated with network arcs rather than network nodes. The application addressed in [54] is to sanitation vehicle routing and scheduling. The problem and solution approach have several complicating factors including:

- seed nodes are used in a manner similar to the Generalized Assignment Heuristic described earlier;
- there are multiple vehicle types where each type has a different size/capacity;
- the vehicles are domiciled at a common depot but during the course of the day the vehicle must make multiple trips to a landfill to dump the refuse that has been collected.

As illustrated in Fig. 6, once an initial feasible solution is obtained, the basic strategy employed is to select a set of routes and then to resolve the problem over the demand set of that set of routes optimally using an MP approach. The MP-model can be called with various decision space options. In particular, the seed-node-to-route assignment can be fixed or variable, the fleet mix can be fixed or variable and the number of trips to the landfill can be fixed or variable (fixing this variable effectively fixes the daily vehicle load). When more variables are fixed, larger problems can be solved but a smaller set of routes can be included. In [54], the subsets of routes chosen was always geographically contiguous. In some cases a metric, which measured solution quality, was employed; the subset chosen had a low value of the metric. In addition, the tool was embedded within a decision support system and the user was allowed to choose the subset of routes to be optimized. This approach was shown to be very effective at generating substantial improvements over a well-accepted heuristic.

In [55], this approach was applied to a multi-commodity flow model that addressed the routing of aircraft for an on-demand air service. The authors compared several strategies for choosing the column set that defined the set of rows to optimize over. They investigated the tradeoff between choosing a smaller set of columns, which led to a smaller number of rows and faster solution times for SP[2] vs a larger set of columns which led to longer solution times for SP[2] but higher quality solutions. For smaller column sets there was less improvement per iteration but each iteration took less time. When column sets were chosen randomly the strategy of having small column sets was superior but when metrics were designed to choose column sets strategically it was better to choose larger column sets. The metrics had objectives similar to those described for the prior application, i.e. one sought to find "low quality" routes and also a set of routes that were "close together".

One can consider a wide range of strategies for defining the column set J'. In analyzing the split delivering vehicle routing problem Archetti et al. [56], employ tabu search to find a region of the solution space that is likely to generate high quality solutions. Specifically, the identify pairs and nodes (and more generally sets) that are likely to be on the same routes. These are then used as the basis for defining the subset of rows to optimize over. In fact, in this cases the search procedure is only loosely aligned with a specific feasible solution. Bent and Van Hentenryck [57] also apply this general approach to a VRP. To define the column set to remove they start by randomly choosing one customer (column) and then augment this customer based on a "relatedness criterion". They use a custom branch-and-bound algorithm to solve the resultant subproblem created.

3.2.2. Restricted Column Set

The Restricted Column Set approach is distinguished by its conceptual simplicity and generality. For literally any mathematical program one could take the columns corresponding to a feasible solution, augment that set by some additional columns, then solve a mathematical program restricted to that column set. The key to successfully applying this idea is the manner in which the additional columns are chosen, which goes hand-in-hand with the exact method used to solve the problem instance created. One approach is simply to generate relatively small and/or tightly constrained problem instances so that general purpose solvers can find a solution rapidly. This is the philosophy employed in the parallel set partitioning algorithm given in Linderoth et al. [58], who, in the course of a more complex procedure, create many compact problems instances that are solved by a general purpose set partitioning algorithm. We note that, using parallelization, the authors are able

to apply this approach many times and choose the best overall solution generated. Of course, the Restricted Column Set approach, as outlined is exactly the basis for column generation approaches to mathematical programs, which can be used iteratively to find exact solutions. In the next section, we discuss how column generation methods are also used in a restricted way to generate approximate solutions. Below, we discuss some more specialized approaches.

General use of solvable special case

Consider a general combinatorial optimization problem defined on a set of elements $N = \{1, \ldots, n\}$ where c_j is the cost of $j \in N$. Let $\Omega = \{S \in N : S \text{ is a feasible solution}\}$, i.e. the optimization problem of interest is: $P = \{\text{Min } \sum_{j \in S} c_j : S \in \Omega\}$. Now suppose that a solvable special case of P is know and characterized by Γ, a family of subsets of N. That is, the optimization problem $\hat{P}(N') = \{\text{Min } \sum_{j \in S} c_j : S \in \Omega \text{ and } j \in N'\}$ can be solved in polynomial time for any $N' \in \Gamma$. Given any such solvable special case, an improvement heuristic is in concept possible. A key ingredient is the ability to augment any feasible solution to obtain an $N' \in \Gamma$. That is, we require the procedure:

Augment(S)
Input: a feasible solution $S \in \Omega$
Output: an $N' \in \Gamma$ with $j \subset N'$.

We can then define the following improvement heuristic:

S-Case-Imp(S)
Set $N' = \text{Augment}(S)$.
Find $S' = \text{argmin}\{\sum_{j \in S} c_j : S \in \Omega \text{ and } S \in N'\}$
Output: S'.

Ahuja et al., [39] illustrate this approach using the TSP and Halin graphs. A Halin graph is an undirected graph obtained by embedding a tree with no nodes of degree 2 in the plane and then connecting all leaf nodes in a cycle so that the graph remains planer. Cornuejols et al. [59] give an $O(n)$ algorithm to solve the TSP on Halin graphs. To apply the procedure given above to this case, Augment would take as input a TSP tour, S, and output a Halin Graph N' that contained S. The new TSP tour output, S' would in general contain arcs from S but also arcs from N that were not contained in S. To our knowledge, this approach has not been tested empirically. It does not appear that this general approach has been broadly explored, however, there would certainly seem to be many possibilities as many problems, particularly network problems are known to be polynomially solvable over restricted problem classes. Examples, include problems restricted to series–parallel networks, acyclic networks or bipartite networks.

The best solution in the union of two solutions

In certain solution approaches many different feasible (and infeasible) solutions are generated. An appealing concept is to look at pairs of such solutions and to find the best solution "within" the pair. This is, in fact reminiscent of genetic algorithms, which employ a "crossover" operation on pairs of "parent" solutions (see [60]). In the classic genetic algorithm crossover operation, characteristics are randomly chosen from the two parent solutions. Alternatively, those characteristics could be chosen optimally [61] (this is perhaps a less controversial form of genetic engineering!!). This technique was applied very effectively in [62]. The problem addressed was to find a minimum cost perfect matching problem in an undirected graph subject to a single generalized upper bound side constraint. The side constraint could be characterized by coloring a special set of edges red. A feasible solution is a perfect matching with no more than b red arcs. The authors dualized the side constraint and iteratively solved the associated Lagrangian relaxation with varying multiplier values. Each time the Lagrangian relaxation was solved a new perfect matching was generated: some of the matchings were feasible (number red arcs $\leq b$) and some were infeasible (number of red arcs $> b$). At the end of the dual ascent phase, the best feasible matching was found in the union of each pair of matchings previously generated.

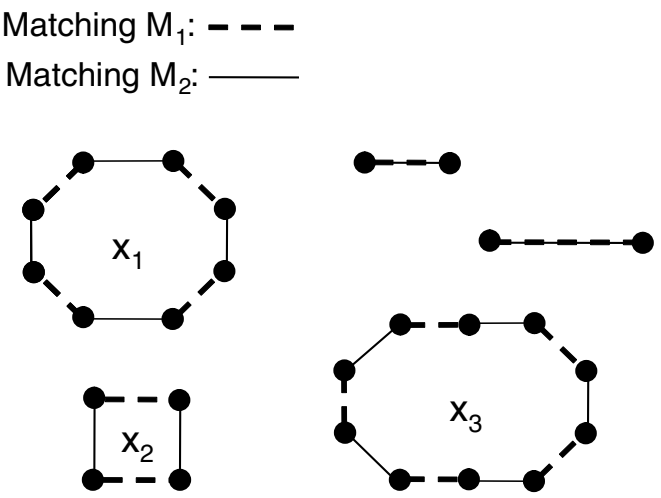

Fig. 7. Union of two perfect matchings with cycle variables.

The union of a pair of perfect matchings (see Fig. 7) is a set of isolated arcs and a set of even length cycles. Each even length cycle contains exactly two matchings that touch all nodes in the cycle. To find a perfect matching for the entire network, one can define an optimization model by associating 0/1 variables with each cycle. For each cycle c we denote by $M_1(c)$ and $M_2(c)$ the two matchings it decomposes into. Let $R(M)$ be the number of red edges in a matching M and $C(M)$ be the cost of a matching M. Suppose without loss of generality that $R(M_1(c)) \geq R(M_2(c))$; now let $r_c = R(M_1(c)) - R(M_2(c))$, $e_c = C(M_1(c)) - C(M_2(c))$ and $b' = b - \sum_c R(M_2(c))$. Then, if we define the decision variable x_c to be 1 if $M_1(c)$ is chosen and 0 if $M_2(c)$ is chosen, we have the following knapsack problem:

$$\text{Min:} \quad \sum_c e_c x_c$$

$$\text{s.t.} \quad \sum_c r_c x_c \leq b'$$

$$x_c \in \{0, 1\} \text{ for all } c.$$

The x_c variables choose one matching for each cycle so that the union of all matchings yields the best (constrained) perfect matching within the restricted network. This is a knapsack problem that can be efficiently solved since the right hand side (b') is bounded by the number of nodes in the original graph.

Aggarwal et al. [63] also investigate optimizing over a decision space defined by the union of two solutions. The core problem investigated is the *independent set* problem and the problem of optimizing over the union of two solutions can be modeled as a bipartite matching problem. The interesting general question is to determine when a particular NP-hard problem can be efficiently solved when the feasible set is restricted to the union of two solutions.

3.3. Parallel savings heuristics

One of the earliest heuristics formally described and analyzed is the so-called savings heuristic for the vehicle routing problem due to Clarke and Wright [64]. The heuristic starts by defining an initial feasible solution to the VRP as a set of single node routes and then iteratively combining pairs of routes that produce a savings in the sense that the cost of the combined route is less than the sum of the costs of the individual routes. This certainly cannot be interpreted as a MP-based approach, however, it is possible to define "parallel" versions of this approach that find a set of such combinations that can be executed simultaneously by solving a matching problem. In

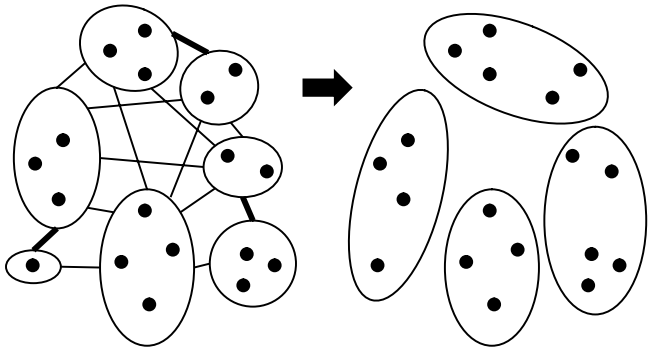

Fig. 8. Parallel savings construction heuristic.

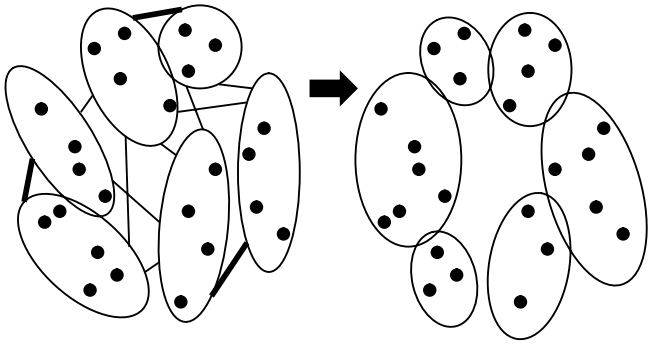

Fig. 9. Parallel savings improvement heuristic.

a more general context, we can apply such methods to problems that have an underlying clustering structure. That is, a route can be viewed as a cluster of nodes, while other examples, include customers associated with particular warehouses, trips performed by a single crew or vehicle, etc. For the generic problem we start with an input set T and define a cost function $c(S)$ for and subset $S \subset T$. The problem is to partition T into subsets, $S_1, S_2, \ldots, S_k$ such that the sum $c(S_1) + c(S_2) + \cdots + c(S_k)$ is minimized. It should be easy to see how the basic Clarke and Wright procedure could be applied in this setting by starting with all cardinality one subsets and then iteratively combining subsets that had a positive savings in a greedy fashion. Alternatively, parallel versions of such approaches can be defined based on the iterative solution of matching problems. Given any feasible solution, $S_1, S_2, \ldots, S_k$, the savings associated with any pair $\{S_i, S_j\}$ can be defined by: $c(S_i) + c(S_j) - c(S_i \cup S_j)$.

As is illustrated in Fig. 8, a matching network can be defined where a node is defined for each set S_i and the weight of the undirected arc between two nodes is the corresponding savings. A maximum weight matching yields a set of pairs of sets that can be simultaneously combined in a way the produces the maximum overall savings. This process can be iterated until no further savings is possible, i.e. the solution to the maximum weight matching problem is the empty set. This process is probably most accurately classified as a construction heuristic since it starts with a trivial solution and iteratively builds up a complex solution.

As illustrated in Fig. 9, this basic approach can be modified to generate an approach that can be naturally viewed as an improvement algorithm. Again for any feasible solution we consider an operation on a pair of subsets, S_i and S_j, where the subsets are combined and the best (or a better) partition of their union $\{S_i^*, S_j^*\}$, is found. Note that finding such an S^* is an example the problem analyzed in the previous section. The savings produced by this operation is: $c(S_i) + c(S_j) - c(S_i^*) - c(S_j^*)$ Using this definition of savings we could apply the same matching based approach defined earlier.

This general approach has been applied both to the vehicle routing problem [65] and to transit crew scheduling [31], which defined and made use of both the construction and improvement versions. A particularly popular variant is the so-called "sequential matching" or "sequential assignment" approach (see Fig. 10), which iteratively builds up a sequence of tasks, where each task has start and end times and (usually) also start and end locations. The unconstrained version of this problem can be solved using network flows (see [22] or [11]) but heuristics approaches have been applied when the sequences are subject to constraints [66–68]. We note that it is usually not the case that the general parallel savings approach can be applied in "direct" fashion; rather a certain amount of cost structuring is required to insure that certain anomalies do not result.

3.4. Math programming based tabu search

Tabu search and, more generally meta-heuristics, has evolved into a very vibrant research area that offers effective solution strategies for a wide range of problems. It is only natural to expect that there should be opportunities for meta-heuristics to make use of mathematical programming. Specific examples of this have been described at other locations in his paper. Here we highlight an approach due to Crainic et al. [69], which is distinctive in the close integration of linear programming basis exchange with a tabu search strategy. The problem considered is the network design problem, which can be described in terms of a set of 0/1 arc variables $\{y_a\}$ and a set of continuous path flow variables $\{h_p\}$. The problem is to choose a set of arcs to open as indicated by the y variables so as to support demand for flow between certain origin–destination node pairs. The flow paths used are designated by the h variables. The approach involves iteratively generating path flows, $\tilde{h}$, that satisfy the demand requirements and then mapping each of these to a y variable solution by:

$$y(\tilde{h})_a = \begin{cases} 1 & \text{if } \sum_{p:a\in p} h_p > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The authors note that, if an optimal solution exists, there is always an extreme point solution to the path variable LP that will map to an optimal (y, h) solution. Thus, they propose the general strategy of enumerating extreme point solutions to the path LP, mapping these to solutions (y, h) and choosing the best such solution. Of course, a total enumeration of this type would be too time consuming. However, they define a procedure for using tabu search [70], for iterating from one path solution to another. Such an iteration involves a standard LP basis exchange operation. In this case, the tabu control structure determines the entering non-basic variable. Additional complexity is involved related to the possibility of column generation and also certain moves involving multiple pivots.

There are potentially many other ways in which MP can be combined with tabu search. For example, Easwaran and Üster [71] use tabu search to improve Benders primal bounds and Pedersen et al. [72] and Gendreau et al. [73] initialize tabu search by rounding a related LP. Glover and Laguna [70] provide an overview of general strategies for combining mathematical programming and tabu search.

4. Using mathematical programming algorithms to generate approximate solutions

It is typically the case that a complete mathematical programming solution package has a large amount of associated "machinery". This is particularly true for mixed integer programming solvers, which employ one or more core linear programming

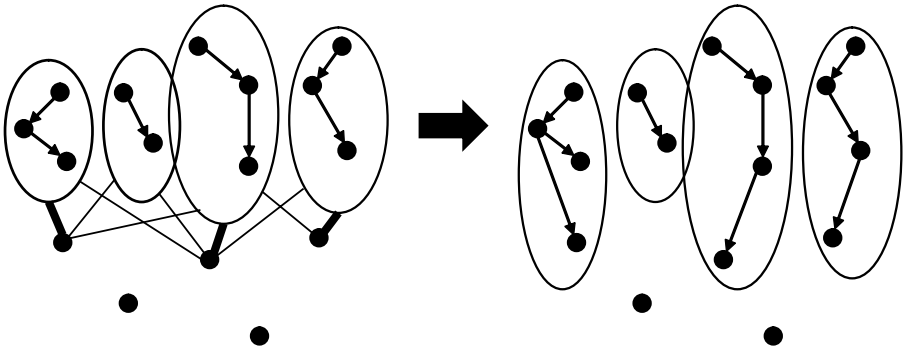

Fig. 10. Sequential matching heuristic.

solvers, together with a branch-and-bound enumeration framework, a suite of preprocessing tools and possibly row and column generation functionality. The complete package represents a powerful set of problem solving capabilities that are engineered to create the final solver. The philosophy underlying the techniques in this section is that one should not be too "pure" is using these capabilities. That is, rather than only using such solvers in a mode that leads to provably optimal solutions, why not use the mathematical programming tools in a, perhaps ad hoc, way to create good solutions? We note that one class of heuristics that falls within this definition are heuristics that modify the output of the solution to a relaxation. We view such heuristics as a special category and treat them in the next section.

4.1. Use of branch-and-bound tolerance

The heart of any branch-and-bound algorithm is the ability to prune an enumeration tree based on a bound on the optimal solution. Assuming a mathematical programming problem with a minimization objective function, at some (or all) nodes, *nd*, in the tree a relaxation, $R(nd)$, is solved which produces a lower bound on the value of the optimal solution, $v_{R(nd)}$. This lower bound applies to all feasible solutions within the subtree rooted at the node in question. Also maintained throughout the algorithm is a best feasible solution, $\hat{F}$, with value $v_{\hat{F}}$. The basic test made at the nodes in the tree is:

if $v_{\hat{F}} \leq v_{R(nd)}$ then prune tree rooted at *nd*.

Since all solutions contained in the tree rooted at *nd* have objective function values greater than or equal to $v_{R(nd)}$, no solutions "better than" $\hat{F}$ are eliminated by this test. This test can be loosened to allow for earlier pruning and thus faster problem solution. Given an absolute tolerance Δ or a relative tolerance ρ the pruning test could be modified to:

if $v_{\hat{F}} \leq v_{R(nd)} + \Delta$ then prune tree rooted at *nd*,
or
if $v_{\hat{F}} \leq v_{R(nd)} * (1 + \rho)$ then prune tree rooted at *nd*.

In this way approximate solutions within an absolute or relative percentage of the optimal can be obtained. Virtually all branch-and-bound solvers have such capabilities.

4.2. Diving heuristics

Probably the simplest heuristic use of a branch-and-bound solver is to stop the progress of the algorithm when the first feasible solution is encountered. If this is coupled with a depth first tree exploration then the solver's machinery is essentially geared toward finding a feasible solution as quickly as possible. Assuming, as is usually the case, that a relaxation is solved at the root node, this strategy will result in a feasible solution with a bound on its deviation from optimality, but the deviation between the solution's value and the bound could, in general, be quite large. This basic depth first strategy has been modified and enhanced in a variety of ways, leading to so-called "diving" or "plunging" heuristics (see [20,74–76] for examples). Here are some of the techniques that have been proposed:

Heuristic variable fixing: whenever an LP relaxation is solved, if the LP assigns a variable an integer value then that variable is fixed at the integer value during any further exploration within the subtree. Note that it is possible that variables that are integer in the LP could become fractional in a subsequent LP without such a restriction.

Rounding: at any node in the tree one or more fractional variables could be set to integer values. The most natural approach to carrying this out would be to round variables that are close to integer values. For example, in [76], within a branch-and-bound tree, whenever an LP relaxation is solved, any 0/1 variable whose LP value is $\geq .9$ is set to 1 and any basic 0/1 variable whose LP value is $\leq .05$ is set to 0.

Iterative column generation: for IP's addressed using column generation, the branch-and-bound process can start by solving the initial LP using a partial (heuristic) column generation; subsequently, additional columns are generated based on heuristic criteria. For example, the technique used by Grötschel et al. [20], which was originally described by Marsten [77], who named it BANG – Branch-ANd-Generate –, defines a "trust region" around the value of the initial LP. Whenever the value of an LP relaxation solved falls outside of the trust region, then additional columns are generated. If this does not sufficiently improve the LP value then a new trust region is defined.

4.3. Beam search

While diving heuristics are based on a depth first search, *beam search* can be viewed as a controlled, heuristic breadth first search. Beam search keeps the size of the branch-and-bound tree manageable by simply pruning nodes based on heuristic criteria. It defines a set of *elite nodes* and explores these while discarding the others. The beam width is the number nodes explored at each level. A variety of heuristic criteria can be considered to define the elite nodes. This approach was introduced in the context of scheduling in [78]. See [79] for a more recent application.

4.4. Variable fixing

Two effective tools used in mixed integer programming (MIP) solvers are variable probing and reduced cost fixing. Both are (exact) techniques for fixing 0/1 variables to a value of 0 or 1 (versions exist for more general IPs as well). Probing temporarily fixes a variable to one of its bounds and then determines the

implication of this on problem feasibility. If setting the variable to 1 implies that the problem becomes infeasible then it can be inferred that the variable can be fixed at 0. The equivalent implication holds relative to setting the variable to 0. For example, suppose that 4 0/1 variables, $\{x_i\}_{i=1}^4$ appear in the following two constraints (possibly, in addition to many others):

$$5x_1 + 3x_2 + 8x_3 + 6x_4 \geq 10$$
$$x_1 + x_3 + x_4 \leq 1.$$

Consider a probe that temporarily sets $x_1 = 1$. Based on the second constraint we see that $x_3 = x_4 = 0$. The first constraint then becomes $3x_2 \geq 5$. This constraint clearly cannot be satisfied so an infeasible problem results. Thus, we can infer that x_1 must be set to 0. Of course, general probing can be quite complex and time consuming, but also very effective in simplifying LP relaxations. More detailed enumerates of probing can be found in [80,75,81].

Reduced cost fixing sets variables in a similar way, however, the criterion used is based on objective function properties. For example, consider an LP relaxation with a minimization objective function. Suppose a non-basic 0/1 variable, x_j had reduced cost $\bar{c}_j$ and the value of the corresponding LP relaxation was z^* and suppose that a feasible solution with value $\hat{z}$ were known, then if $z^* + c_j > \hat{z}$, we can conclude that x_j can be permanently fixed to 0. In this case, the reason is that any solution with $x_j = 1$ would have a value worse than a known feasible solution. Heuristic solution strategies have been devised based both on (1) embedding variable fixing of this type within iterative schemes that involve non-optimality preserving steps and (2) approximate versions of the above conditions.

We now describe an approach (the ANS Heuristic) due to Atamturk et al. [82] for solving the set partitioning problem that employs variable fixing within a more complex solution process (see also [58] for a parallel version). The solution process employs two other set partitioning problem reduction techniques, which we now describe:

Duplicate column elimination: whenever two identical columns exist, the one with the higher cost can be eliminated;
Row domination: if $T(i)$ is the set of columns that contain a 1 in row i, then whenever $T(i)$ strictly contains $T(k)$, we say k dominates i and row i can be deleted and all variable appearing in $T(i) - T(k)$ can be set to 0. For example, consider the set partitioning constraints: $x_1 + x_2 = 1$ and $x_1 + x_2 + x_3 + x_4 = 1$. Since the first constraint implies x_1 or x_2 must be 1, the second then implies $x_3 = x_4 = 0$ and furthermore, the second is redundant in light of the first.

ANS heuristic

1. Remove duplicate columns.
2. Apply row dominance and variable probing.
3. Solve LP relaxation. If LP is integral then exit with optimal solution.
4. Apply primal heuristic to find feasible (integer) solution.
5. Apply reduced cost variable fixing; if any new variables are fixed then go to step (1).
6. Generate valid inequalities; if new valid inequalities found then go to step (3).
7. Terminate and report best feasible solution.

This procedure actually employs nearly a complete arsenal of branch-and-bound technology and, in fact, resembles a branch-and-bound algorithm. However, there never is any variable branching. Rather, iterations occur based on additional variable fixing (step (5)) or based on the identification of new valid inequalities (step (6)). We note that as more variables are fixed, it becomes more likely the row dominance will be successful and as the LP relaxation becomes stronger and the heuristic solution better, it becomes more likely that variables can be fixed. The variable fixing carried out is exact in the sense that any variables fixed to 0 or 1 have that value in an optimal solution. In the next section we present a Lagrangian relaxation based heuristic that iteratively fixes variables using heuristic criteria.

4.5. Partial column generation

It is very often (possibly almost always) the case that column generators do not truly consider all feasible columns. That is parameters are set so that only "reasonable" columns can be generated. In addition, to reduce the computational burden, column generation can be stopped based on a variety of heuristic criterion. A further step in the direction of heuristic column generation is to purposely only generate "good" columns. For example, Kelly and Xu [83] describe an approach in which simple heuristics for the VRP are executed multiple times. The end product of this step is not taken to be a single solution but rather the union of all the solutions generated. The set of routes in this union is then used as the input column set for a set partitioning step to generate the final solution. It should be noted that this author has heard practitioners remark that, sometimes, the trick to finding a great solution is to find one or two "really bad looking" columns that make the overall solution work. Thus, strategies that focus only on a subset of "good" columns can potentially lead to poor results. Note that other heuristic partial column generation schemes were discussed in Section 4.2 in the context of partial column generation.

5. Relaxation based approaches

5.1. Rounding the solution to an LP

The simplest and, perhaps most tempting approach, to an optimization based heuristic is to round the solution to a linear programming relaxation. Certainly this approach is used extensively in practice. In fact, it is probably the case that most linear programming applications are in reality mixed integer programming applications, where from a practical standpoint, rounding an LP solution is a natural, reasonable approach that introduces little error. For example, there is little need for analysis to make a decision to round a solution to produce 1123.3 widgets to a solution to produce 1123. On the other hand, rounding a fractional solution to 0/1 problem has the potential for introducing more error and, in fact, finding a feasible integer solution from a corresponding fractional solution can be quite challenging. Nonetheless, there are cases where solutions to the LP relaxation of 0/1 problems can provide a very effective start to creating a good feasible 0/1 solution. We start with a discussion of the *Minimum Weight Node Cover Problem* (MWNC), which serves to illustrate both potential good and bad scenarios that can occur. Given an undirected network, $G = (N, A)$, a node cover is a subset of nodes that touches each arc at least once. If we define node weights c_i for all $i \in N$, then the minimum weight node cover problem can be formulated as:

$$\text{MWNC: Min: } \sum_i c_i x_i$$
$$\text{s.t. } x_i + x_j \geq 1 \text{ for all } (i,j) \in A$$
$$0 \leq x_i \leq 1 \text{ and integer for all } i \in N.$$

MWNC would seem to be particularly amenable to a rounding approach due to the structure of its constraints. For any arc (i, j) either x_i or x_j must be one. Moreover it is clear that if we let $\{x_i^*\}$ be a solution to the LP relaxation of MWNC, then for any arc (i, j) either $x_i^* \geq .5$ or $x_j^* \geq .5$. This implies that if we round up any x_i^* that has a value of at least.5 and set the other variables to 0, then a feasible solution will result. It is also the case that this rounding operation cannot increase the value of the objective function by more than a factor of 2. Thus, if z^*_{MWNC} is the value of the LP

relaxation and $\hat{z}_{MWNC}$ is the value of the feasible solution obtained by rounding then $\hat{z}_{MWNC} \leq 2z^*_{MWNC}$ (see [84] and also [85] pages 66–67). This in turn implies that the solution obtained by rounding is within a factor of 2 of the optimal solution. Furthermore, MWNC has the property that any variable x_i that is 1 in a solution to the LP relaxation is also 1 in an optimal IP solution [86]. All of these properties seem to point to the very effective use of rounding. On the other hand, it is also the case, that MWNC has the so-called half-integer property, which states that, in any extreme point solution to the LP relaxation, all variables take on values of 0, 1 or 1/2. While the rounding analysis given above might have seemed very attractive in cases where there was a mix of relatively high and low fractional values, it become less appealing in cases where many variable have a value of 1/2. In fact, Pulleyblank [87] shows that, for the cardinality node cover problem, where $c_i = 1$ for all i, the optimal solution to the LP relaxation has $x_i = 1/2$ for all i if and only if the underlying graph has a property called nontrivial 2-bicritical. He then shows that asymptotically the probability that all graphs possess this property approaches 1, i.e. for almost all graphs the LP relaxation has $x_i = 1/2$ for all i. Of course, this does not mean that there are not important cases, e.g. sparse graphs and certain instances of the weighted (not all $c_i = 1$) problem, where solutions with many 0/1 values are not more common. In fact, these properties have served as the basis for effective heuristics in certain specific contexts (see for example, [88,89]). This "case study" suggests several questions related to the use of rounding:

What thresholds should be used for rounding?
What is the maximum (or average) error introduced by rounding?
What is the likelihood that a large number variables will be 1 in a "typical" LP solution?
What is the likelihood that a large number of variable values will be "close to" 0 or 1 in a "typical" solution?

There is not a general theory that addresses these questions. Rather, specific individual problems must be considered on a case by case basis, in a way that takes these issues into account.

We now describe a general approach to rounding that can be applied to certain classes of MIPs. It is often the case in MIPs that 0/1 variables are associated with the opening of resources used to accomplish some objective. For example, in the network design problem, 0/1 variables are associated with the purchase or installation of link capacity, the fixed resource, to be used to route traffic, which determines the operational costs. Thus, one needs to trade off investments in fixed resources with the resultant operational costs. In such cases, once the 0/1 variables are set, determining the fixed resources, then an LP can be solved to determine feasibility and the associated operational costs. This suggests an iterative threshold approach to rounding, such as the one applied successfully in [90]. The approach starts by solving an LP relaxation; it then searches for an appropriate threshold to determine which of the fractional 0/1 variables should be rounded to 1. We can interpret the threshold as measure of the fixed resource investment level. Thus, the role of the solution to the LP relaxation is to map the fixed resource investment level into a specific set of resources (links) to be used.

To define the details of this approach, we represent a generic MIP as:

$$\begin{aligned} \text{PR: Min: } & cx + dy \\ \text{s.t. } & A(x, y) = b \\ & x \geq 0, y \in \{0, 1\}^n. \end{aligned}$$

Let y^* be a y-vector resulting from solving the LP relaxation to PR and $z^*_{PR}(y')$ be the value of the LP the results when y is set equal to y'. Further, let $\{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_r\}$ be a set of threshold values with $0 < \hat{t}_k < 1$ for all k. We can now define the following rounding heuristic:

Iterative threshold heuristic:

1. Solve the LP relaxation of PR generating y^*.
2. For $k = 1, r$:
   For all i, set $y'_i = 1$ if $y'_i \geq \hat{t}_k$, set $y_i = 0$, otherwise.
   Fix $y = y'$ and solve the LP, i.e. find $\hat{z}_k = z^*(y')$
3. Choose $\text{Min}_k \hat{z}_k$ and output corresponding solution to PR.

We note that it could easily be the case that some of the LPs solved in step (2) could be infeasible, in which case $\hat{z}_k$ would be set to a large constant. In concept this approach could be applied to any MIP, but the appropriateness of doing so will depend on the problem specifics. It is interesting to contrast this approach with the tabu search method described in Section 3.4. Both methods address network design problems. The tabu search approach finds multiple continuous variable (routing) solutions and maps each of these to a 0/1 solution, whereas this approach, which is perhaps more typical, finds multiple 0/1 variable solutions and maps these to continuous variable solutions. Many applications of rounding, e.g. especially when carried out in the context of branch-and-bound, round multiple LP solutions and choose amongst the feasible integer solutions generated.

It is natural to consider employing randomization in this process. Glover and Laguna ([70], Chapter 6) describe the concept of *directional rounding*. It has the attractive property that it can be iteratively applied within tabu search or other randomization schemes to generate sequence of possible solutions. Starting with x^*, the fractional solution to the LP relaxation of a 0/1 problem, and any other $0 \leq x \leq 1$, which may or may not be integer, define the 0/1 vector $\delta(x^*)$ by:

$$\begin{aligned} &\delta(x^*)_i = 1 \quad \text{if } x^*_i < x_i \\ &\delta(x^*)_i = 0 \quad \text{if } x^*_i > x_i \\ &\delta(x^*)_i = 0 \text{ or } 1 \quad \text{if } x^*_i = x_i. \end{aligned}$$

The authors propose approaches where x is modified based on alternate schemes that employ randomization. For each value of x, $\delta(x^*)$ is recomputed creating a sequence of heuristic solutions. Of course, an organized search or randomization strategy could also be applied to generate alternate values for x.

5.2. Searching around an LP optimum

Simple rounding can be viewed as a process that maps an LP optimum into a single (perhaps arbitrary) near-by integer solution. Alternatively, it is probably natural to consider heuristic strategies for finding good feasible solutions to IPs by enumerating around the LP optimum. Yet, there is surprisingly little formal work in this area. An exception is the appealing work of Balas et al., [91]. They propose the OCTANE (OCTAhedral Neighborhood Enumeration) heuristic which enumerates 0–1 solutions in the vicinity of the optimal solution to the LP relaxation of a 0–1 IP. To understand this heuristic we first consider the following polyhedron defined around the origin: $\text{PH} = \{x_i : \delta_i x_i \leq n/2 \text{ for all } \delta \in \{\pm 1\}^n\}$. Note that there are 2^n constraints, one for each ± 1 vector in $\Re^n$. The heuristic starts by translating an LP solution, x, by subtracting 1/2 from each component. This translation will always place x in the interior of PH. Then, a direction away from x is chosen and facets (equivalently constraints) of PH are enumerated in the order in which they are encountered. Each facet enumerated is mapped to a unique 0/1 solution. All such solutions are tested for feasibility; the heuristic outputs the best such feasible solution. The solution $\tilde{x}(\delta)$ associated with each facet of PH with associated constraint vector δ is defined by:

$$\begin{aligned} \tilde{x}(\delta) = 1 \quad & \text{if } \delta_i = 1 \\ 0 \quad & \text{if } \delta_i = -1. \end{aligned}$$

A formal statement of the heuristic is given below.

OCTANE:

1. Let x be the fractional solution to the LP relaxation of a 0–1 IP. Transform x to $\hat{x}$ by $\hat{x}_i = x_i - 1/2$.
2. Choose a vector $a \in \Re^n$ and consider the half line $r = \{\hat{x} + \lambda a : \lambda \geq 0\}$.
3. Find $\{\delta^1, \delta^2, \ldots, \delta^k\}$, the first k facets of PH intersected by r and determine the corresponding 0–1 solutions: $\{\tilde{x}(\delta^1), \tilde{x}(\delta^2), \ldots, \tilde{x}(\delta^k)\}$.
4. The points in $\{\tilde{x}(\delta^1), \tilde{x}(\delta^2), \ldots, \tilde{x}(\delta^k)\}$ that are feasible serve as heuristic solutions.

A key aspect of this approach is that the enumeration in step 3 can be carried out efficiently. An important parameter is the choice of the direction a, which can be difficult to set effectively. The authors experimented with several variants. They also embedded this approach within a branch and cut framework, which effectively allowed for the starting point (x) to be varied. Probably the biggest potential shortcoming of this approach is difficulty in identifying feasible solutions. Glover and Laguna (see [70], Chapter 6) describe a framework called cut search for similar approaches. The technique extends the edges of the LP cone to identify a hyper-rectangle to be searched for integer solutions. In addition to enumerating solutions in a particular order, they suggest optimizing over the hyper-rectangle, which usually is not too difficult because it is relatively small in size.

5.3. Other primal approaches

Another class of heuristics creates a feasible integer solution to an integer program based on information gained from the primal solution to an LP relaxation. However, these approaches are not based on rounding but rather on prioritization schemes based on information contained in the primal solution. Such approaches are most extensively analyzed in the field of machine scheduling (see for example [92]). The problem of scheduling a single machine to minimize total weighted completion time is one of the most basic problems in scheduling theory. The problem is defined by specifying a set of n jobs where each job j has a positive weight w_j and a non-negative processing time p_j. For any feasible schedule we can define for each job a completion time C_j. A solution to the problem is a schedule which minimizes $\sum_j C_j$ or equivalently, $\sum_j C_j/n$. The unconstrained version of this problem can be solved in polynomial time; however, most interesting constrained versions are NP-hard. Of particular note is the case where each job j has a *release time* r_j before which the job cannot be scheduled and the case, where precedence constraints, of the form $j \prec k$, are specified whenever job j is constrained to be completed before job k. We now define two classes of LPs for these problems. The time indexed formulations are based on variables, x_{jt}, which are defined to be 1 if job j is completed in time period t. Here time is assumed to be discretized into time intervals, $t = 1, 2, \ldots, T$ at which all activities start or end. The *completion time formulations* directly employ completion time variables C_j as defined above. Both of these are defined for the problem with release times, however, variants exist for the precedence constrained problem as well. To enforce the release time constraints, x_{jt} is defined only for $t = r_j - p_j + 1, \ldots, T$

Time indexed formulation

$$\text{Min:} \quad \sum_{jt} c_{jt} x_{jt}$$

$$\text{s.t.} \quad \sum_{t=r_j+p_j}^{T} x_{jt} = 1 \quad \text{for all } j$$

$$\sum_{j=1}^{n} \sum_{s=1}^{p_j+1} x_{st} \leq 1 \quad \text{for all } t$$

$$x_{jt} \in \{0, 1\} \quad \text{for all } j \text{ and } t.$$

Here, $c_{jt} = w_{jt}$.

The completion time formulations use valid inequalities developed by Queyranne [93] and Wolsey [94]. We define the set of jobs as $N = \{1, 2, \ldots, n\}$ and any $S \subseteq N$:

$$p(S) = \sum_{j \in S} p_j$$

$$p^2(S) = \sum_{j \in S} p_j^2$$

$$r_{min}(S) = \min_{j \in S} r_j.$$

With these definitions we can now define for any $S \subseteq N$,

$$\rho(S) = r_{min}(S)p(S)1/2(p^2(S) + p(S)^2)$$

and state the formulation:

Completion time formulation

$$\text{Min:} \quad \sum_j w_j C_j$$

$$\text{s.t.} \quad \sum_{j \in S} p_j C_j \geq \rho(S) \quad \text{for all } S \subseteq N$$

$$C_j \geq 0 \quad \text{for all } j.$$

In general this formulation does not produce feasible completion times. Rather its value is a lower bound on the value of the corresponding scheduling problem and there is not immediate corresponding integer program that gives a feasible schedule. The time indexed formulations and the associated algorithms are pseudo-polynomial in problem size and, in general, not polynomial since the number of variables grows as a function of total processing time. The completion time formulations produce weaker bounds than the LP relaxation of the time indexed formulations but they can be solved in $O(n \log n)$ time by specialized algorithms in spite of the exponential size of the constraint set (see [95]). For the time indexed formulation the completion time is given by $\hat{C}_j = t x_{jt}$. Of course, when the LP relaxation of the time indexed formulation is solved, like the completion time formulation, this expression does not give a set of completion times that is necessarily feasible. Once completion time "estimates" are produced by either relaxation, the following heuristic could be applied.

Schedule by C_j

Order jobs according to increasing value of C_j.

for $j = 1, \ldots, n$: schedule job j at earliest possible time.

This heuristic applied to either formulation is known to have a solution quality within a factor of 3 of the optimal. Furthermore, Savelsbergh et al. [96] provide computational evidence that approaches of this type can perform quite well in practice. Thus, we can view this general approach as using an LP solution to provide job priorities that are used as a basis for scheduling. Such approaches have been used successfully to attack a variety of scheduling problems.

While approaches of this type are most well studied in the scheduling literature, examples of similar techniques can be found elsewhere. For example, in Raidl [97], the relative (fractional) values of variables in an LP relaxation are used as priorities in several places within a genetic algorithms. The more standard approach would choose a random variable order for such operations. In [76], a greedy algorithm is used to assign component types to machine "sections", where the greedy algorithm employs a component type ordering based on variable values from an LP relaxation. In [98] a type of "nearest neighbor" algorithm is executed on a network, where the "closeness" of two nodes is measured relative to the value of arc variable from an LP relaxation. More generally the solutions to LP relaxations have been used to guide subsequent heuristics. Rousseau and his co-authors [28,29] solve an LP and then use it to provide guidance to subsequent

heuristics for forming crew schedules. Fernández et al. [99] use the rounded solution to an LP to define a connectivity structure that guides subsequent heuristics that create a solution to the rural postman problem. Generally, such LP's also generate a bound on the value of an optimal solution.

5.4. Lagrangian relaxation based heuristics

Lagrangian relaxations have been used extensively over the years to construct practical solution approaches to a variety of mathematical programming problems (see [100,101] for background). For an integer program of the form,

$$z^* = \text{Min: } cx \tag{12}$$
$$\text{s.t. } A_1x \geq b_1 \tag{13}$$
$$A_2x \geq b_2 \tag{14}$$
$$x \geq 0 \text{ and integer} \tag{15}$$

a Lagrangian relaxation can be effective, when one constraint set say, (13), is "easy" to handle and another say, (14) is more difficult to handle. In such cases, constraint set (14) can be dualized creating, for a $\lambda \geq 0$ matching the row dimension of A_2, the relaxation:

$$LR(\lambda) = \text{Min: } cx + \lambda(b_2 - A_2x)$$
$$\text{s.t. } A_1x \geq b_1$$
$$x \geq 0 \text{ and integer.}$$

Dual ascent procedures are typically employed to find a value of λ that approximately solves the problem: $Max_{\lambda \geq 0} LR(\lambda)$. This provides a lower bound on z^* that could, for example, be used within a branch-and-bound algorithm.

We wish to highlight here the manner in which heuristics to find good (primal) feasible solutions can be structured in this setting. For the case, where A_2 consists of a single row, it is typically the case that, in the course of a dual ascent procedure, which solves the Lagrangian relaxation for several values of λ, several feasible solutions will be generated, e.g. if a feasible solution exists then, for large enough value of the scalar λ, one will be generated by solving the relaxation. This will also tend to be the case when A_2 contains a relatively small number of rows. In such cases, one or more feasible solutions will be generated and, in addition to choosing the best of these, improvement procedures can be executed to find good feasible solutions (see [62] as well as Section 3.2.2). More generally and, perhaps more typically, no feasible solutions are generated during the dual ascent procedure. However, many heuristics have been structured to create feasible solutions that are "close" to the (infeasible) primal solutions generated. As an example, we consider the network design problem discussed earlier, which seeks a set of network arcs to "open" in order to support traffic flows between a set of origins and destinations with associated flow demands. Given a proposed set of arcs to open a multi-commodity flow problem can be solved to find the appropriate traffic flows. Hellstrand and Holmberg [102] describe a Lagrangian relaxation based approach for the network design problem. Each iteration of their dual ascent outputs a primal solution that specifies a set of arcs to open. This set typically is not feasible in that it does not admit a feasible flow. On the other hand, the authors are able to find good feasible solutions by (1) taking the union of the arcs open on two successive dual ascent iterations, (2) finding a minimum cost feasible routing of traffic (if one exists) and (3) closing any unused arcs. In general the philosophy behind Lagrangian relaxation based heuristics is to find a feasible primal solution that is "close" to the (usually infeasible) primal solution generated by solving the relaxation. There are many other examples of this type; see for example [103–107].

5.5. Primal heuristic based on dual information

Given a dual solution, e.g. obtained from solving an LP, certain primal heuristics employ reduced costs. We illustrate such approaches on the *set covering* problem, which is a modification of the set partitioning problem, SP, obtained by replacing the "=" in (7) with a "$\geq$". Defining a column objective function coefficient by c_j and the set of rows column j covers by P_j the Chvatal heuristic [108] for set covering weights columns by $|P_j|/c_j$ and successively chooses the column with highest value until all rows are covered. This approach has been modified to use LP reduced costs rather than actual costs.

A similar approach has been used where Lagrange multiplier values replace linear programming dual values. For problems in the *integrality property*, an optimal set of Lagrange multipliers are also optimal linear programming dual variables so these approaches would appear to be equivalent. The key difference is that Lagrangian relaxations are rarely solved to optimality. Fast dual ascent procedures are used to find "near-optimal" multipliers much more quickly than the linear program could be solved. Fisher and Kedia [109] were the first to propose the use of "reduced cost" based on Lagrange multipliers in greedy heuristics. For the set covering problem once the one nontrivial constraint set is dualized, the Lagrangian relaxation can be trivially solved. Given a column j and a Lagrangian multiplier vector u^*, a column weight γ_j is defined by:

$$\gamma_j = c_j - \sum_{i \in P_j \cup M^*} u_i^*$$

where M^* is the set of uncovered rows. The associated greedy heuristic successively chooses the column with the smallest γ_j value. After each column choice M^* is updated (but u^* is not).

More recently this basic idea has been embellished and applied in a variety of ways. We note in particular the highly effective heuristic of Caprara et al. [110], which won a contest run by the Italian Railway. Based on their experiments the authors defined a new, more effective column weight, σ_j by

$$\sigma_j = \gamma_j/\mu_j \quad \text{if } \gamma_j > 0$$
$$\sigma_j = \gamma_j\mu_j \quad \text{if } \gamma_j < 0$$

where μ_j is the number of uncovered columns covered by column j.

The authors' computational experience showed that very similar near-optimal multiplier sets, when used to guide a heuristic, can produce very different primal solutions. Thus, they applied the basic heuristic procedure to several different multiplier sets. They also applied the procedure iteratively where, after each iteration, a set of variables was fixed to 1, then the entire procedure re-applied to the reduced problem obtained by considering only the free variables. The overall procedure is given by:

CFT heuristic
repeat until x^* cannot be improved

> *subgradient phase*: find a near-optimal Lagrange multiplier vector u^*
> *heuristic phase*: starting from u^*, generate a sequence of near-optimal multiplier vectors and for each vector generate a feasible solution to SCP (update the best incumbent x^* if appropriate)
> *variable fixing phase*: select a subset of "good" columns and fix to 1 the corresponding variables.

The distinctive aspects of this approach are that several sets of Lagrange multipliers are found and each is used to generate a primal solution. Secondly, the overall process iterates where at each iteration a set of variables is fixed to one. This allows for the sequential generation of new multiplier sets which are "customized" to the subproblem that remains after each iteration.

6. Conclusions

It is clear that MP-models and methods have been applied in a wide array of ways to generate approximate solutions to problems. As was stated in Section 2, when an MP-model is embedded in a real problem setting, whether it is called a heuristic or exact method is a matter of interpretation. Thus, to a degree, any research involving the application of mathematical programming contributes to the study of MP-based heuristics. This paper has emphasized practical aspects of the subject it has covered, yet, it is noteworthy that variety of results form worst case and asymptotic analysis of heuristics as well as underlying mathematical programming theory, have provided many useful insights. In this closing section, we provide a few general conclusions and also suggest some research trends and directions.

The "Bag of Tricks"

A review of this section reveals the recurrent use of several models and techniques. Clearly, it is vital that the heuristic designer know these well as they constitute an essential "bag of tricks". Some core well-solved problems arise in several application settings and represent a fundamental component in many strategies. These include: shortest paths and dynamic programming, assignments, matchings and network flows. Although NP-hard, the set partitioning and set covering models are used in many practical settings and play an important role in solution strategies. Linear programming and Lagrangian relaxation form the basis of many exact and approximate solution strategies, based both on the primal and dual information they provide. Finally, the individual components of mixed integer programming branch-and-bound solvers are frequently broken out and used in variety of ways to create problem solving tools.

Relative performance of MP-based heuristics

One, of course, is tempted to ask: How do MP-based heuristics compared to other heuristics? Which are more effective overall? Which are used more extensively in practice? The designer of an improvement heuristic very often faces a basic tradeoff: a technique that executes many "quick" moves, each of which yields a small improvement vs. a technique that executes fewer, more time-consuming large-improvement moves. Of course, the very significant recent trend is toward meta-heuristics, which (in most cases) introduce a degree of randomization. If one considers the volume of recent research output, it would seem that the non-MP-based approaches are "winning". On the other hand, if one reviews the major application areas, such as transit and airline crew and vehicle scheduling, it seems clear the MP-based approaches dominate. In the author's opinion, this is the test that really counts.

MP-based meta-heuristics

Given the large level activity in the area of meta-heuristics, it is only natural to feel that more attention should be given to MP-based meta-heuristics. This survey includes some examples, but it would seem that more activity in this area is warranted. Meta-heuristics can produce solutions of very high quality for large problems; however, in nearly all cases, they produce no quality guarantee. A challenge at the interface of the two fields is to design meta-heuristics that produce such guarantees, e.g. by searching both the primal and dual spaces.

Iterative variable fixing heuristics

The Diving, ASN and CFT heuristics all employ similar strategies. A basic iteration results in a set of variables being assigned values of 0 or 1. This variable assignment leads to a new mathematical program, which is further analyzed producing new variable assignments, etc. The similarities in these approaches suggest a further analysis into the general structure of such approaches including comparing exact and heuristic alternatives for fixing variables and creating new dual solutions and bounds.

Formal analysis of decomposition strategies

Our presentation in Section 2 of decomposition strategies references results from worst case and asymptotic analysis of heuristics but did not make use of formal decomposition analysis. Of course, Benders and Lagrangian decomposition principles (see e.g. [100]) can be used to provide a framework for certain decomposition heuristics; however, they have played a relatively minor role in much of the research in this area. It would seem that an in-depth analysis of the techniques employed in various MP-based decompositions could yield new general results and frameworks.

Heuristics based on solvable special cases of combinatorial optimization problems

Section 3.2.2 identified a class of improvement heuristics that make direct use of solvable special cases of combinatorial optimization problems. These methods have shown promise but seem to have received relatively little research attention.

Flexible branch-and-bound solvers

Branch-and-bound solvers have achieved high level of effectiveness as well as flexibility. It would appear that this trend should be pushed even further to allow users to create, using parameter settings, broad classes of heuristics of the type described in Section 4.2.

Acknowledgement

Some of this work was carried out while the author was visiting the Institute for Mathematics and its Applications (IMA) at the University of Minnesota. The IMA's support is gratefully acknowledged.

References

[1] M.X. Goemans, D.P. Williamson, The primal–dual method for approximation algorithms and its application to network design problems, in: D. Hochbaum (Ed.), Approximation Algorithms, 1997.
[2] Hansen, P.J. Brimberg, D. Urošević, N. Mladenović, Primal-dual variable neighborhood search for the simple plant-location problem, INFORMS Journal on Computing 19 (2007) 552–564.
[3] W.B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, John Wiley and Sons, New York, 2007.
[4] M.L. Fisher, Vehicle Routing, in: M. Ball, T. Magnanti, C. Monma, G. Nemhauser (Eds.), Handbook of Operations Research and Management Science: Network Routing, Elsevier, Amsterdam, 1995.
[5] G. Frederickson, M. Hecht, C. Kim, Approximation algorithms for some routing problems, SIAM Journal on Computing 7 (1978) 178–193.
[6] N. Christofides, Worst case analysis of a new heuristic for the traveling salesman problem, Technical Report, GSIA, Carnegie-Mellon University Pittsburgh, PA, 1976.
[7] N. Christofides, Vehicle Routing, in: E. Lawler, J.K. Lenstra, A. Rinooy Kan, D. Shmoys (Eds.), The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley and Sons, Eastbourne, UK, 1985, pp. 431–448.
[8] M. Fisher, R. Jaikumar, A generalized assignment heuristic for the vehicle routing problem, Networks 11 (1981) 109–124.
[9] J. Bramel, D. Simchi-Levi, A location based heuristic for general routing problems, Operations Research 43 (1995) 649–660.
[10] J. Bramel, D. Simchi-Levi, Probabilistic analysis and practical algorithms for the vehicle routing problem with time windows, Operations Research 44 (1996) 501–509.
[11] R Ahuja, T. Magnanti, J. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[12] L. Bodin, B. Golden, A. Assad, M. Ball, The state of the art in the routing and scheduling of vehicles and crews, Computers and Operations Research 10 (1983) 63–211.
[13] D. Bienstock, J. Bramel, D. Simchi-Levi, A probabilistic analysis of tour partitioning heuristics for the capacitated vehicle routing problem with unsplit demands, Mathematics of Operations Research 18 (1993) 786–802.
[14] M. Haimovich, A. Rinooy Kan, Bounds and heuristics for capacitated routing problems, Mathematics of Operations Research 10 (1985) 527–542.

[15] C. Barnhart, A. Cohn, Airline schedule planning: accomplishments and opportunities, Manufacturing and Service Operations Management 6 (2003) 3–22.
[16] C. Barnhart, A.M. Cohn, E.L. Johnson, D. Klabjan, G.L. Nemhauser, P.H. Vance, Airline crew scheduling, in: Randolph W. Hall (Ed.), Handbook of Transportation Science, 2nd ed., Kluwer Academic Publishers, Norwell, MA, 2001.
[17] C. Barnhart, E.L. Johnson, G. Nemhauser, P.H. Vance, Crew scheduling, in: Randolph W. Hall (Ed.), Handbook of Transportation Science, Kluwer Academic Publishers, Norwell, MA, 1999, pp. 493–521.
[18] J. Daduna, I. Branco, J. Paixao (Eds.), Computer Aided Transit Scheduling, in: Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 1995.
[19] J. Daduna, A. Wren (Eds.), Computer Aided Transit Scheduling, in: Lecture Notes in Economics and Mathematical Systems, vol. 308, Springer-Verlag, Berlin, 1988.
[20] M. Grötschel, R. Borndörfer, A. Löbel, in: W. Jäger, H.-J. Krebs (Eds.), Duty Scheduling in Public Transit, in: MATHEMATICS—Key Technology for the Future, Springer, 2003, pp. 653–674.
[21] J. Rousseau (Ed.), Computer Scheduling of Public Transport: 2, North Holland, New York, 1985.
[22] G. Dantzig, D. Fulkerson, Minimizing the number of tankers to meet a fixed schedule, Naval Research Logistics Quarterly 1 (1954) 217–222.
[23] D. Bienstock, M. Goemans, D. Simchi-Levi, D. Williamson, A note on the prize collecting traveling salesman problem, Mathematical Programming 59 (1993) 413–420.
[24] B. Rexing, C. Barhhart, T. Kniker, A. Jarrah, N. Krishnamurthy, Airline fleet assignment with time windows, Transportation Science 34 (2000) 1–20.
[25] M.O. Ball, L. Bodin, J. Greenberg, in: J. Rousseau (Ed.), Enhancements to the RUCUSII crew scheduling system, in: computer scheduling of public transport: 2, North Holland, New York, 1985, pp. 279–294.
[26] J. Rousseau, J. Blais, An interactive system for buses and crew scheduling, in: J. Rousseau (Ed.), Computer Scheduling of Public Transport: 2, North Holland, New York, 1985, pp. 473–491.
[27] M.O. Ball, H. Benoit, A lagrangian relaxation based heuristic for the urban transit crew scheduling problem, in: J. Daduna, A. Wren (Eds.), Computer Aided Transit Scheduling of Public Transport: Proceedings of Fourth International Workshop on Computer-Aided Scheduling of Public Transport, in: Springer-Verlag Lecture Notes in Economics and Mathematical Systems, vol. 308, Springer-Verlag, Berlin, 1988, pp. 54–67.
[28] J.-Y. Blais, J. Lamont, J.-M. Rousseau, The HASTUS vehicle and manpower scheduling system at the Societ de transport de la Communaut urbaine de Montral, Interfaces 20 (1990) 26–42.
[29] R. Lessard, J.-M. Rousseau, D. Dupuis, HASTUS I: a mathematical programming approach to the bus crew scheduling problem, in: A. Wren (Ed.), Computer Scheduling of Public Transport, North Holland, 1981, pp. 255–267.
[30] R. Freling, D. Huisman, A. Wagelmans, Models and algorithms for integration of crew and vehicle scheduling, Journal of Scheduling 6 (2003) 63–85.
[31] M.O. Ball, L. Bodin, R. Dial, A matching based heuristic for scheduling mass transit crews and vehicles, Transportation Science 17 (1983) 4–31.
[32] J. Cordeau, G. Stojkociv, F. Soumis, J. Desrosiers, Benders decomposition for simultaneous aircraft routing and crew scheduling, Transportation Science 35 (2001) 375–388.
[33] D. Klabjan, E. Johnson, G. Nemhauser, Airline crew scheduling with time windows and plane count constraints, Transportation Science 36 (2002) 337–348.
[34] A. Cohn, C. Barnhart, Improving crew scheduling by incorporating key maintenance routing decisions, Operations Research 51 (2003) 387–396.
[35] M. Lohatepanont, C. Barnhart, Airline schedule planning: integrated models and algorithms for schedule design and fleet assignment, Transportation Science 38 (2004) 19–32.
[36] R. Sandhu, D. Klabjan, Integrated airline fleeting and crew-pairing decisions, Operations Research 55 (2007) 439–456.
[37] C. Barnhart, A. Farahat, M. Lohatepanont, Airline fleet assignment with enhanced revenue modeling, Operations Research 57 (2009) 231–244.
[38] P. McAree, L. Bodin, M.O. Ball, Models for the design and analysis of a large package sort facility, Networks 39 (2002) 107–120.
[39] R. Ahuja, O. Ergun, J. Orlin, A. Punnen, A survey of very large scale neighborhood search techniques, Discrete Applied Mathematics 123 (2002) 75–102.
[40] C.N. Potts, S.I. van de Velde, Dynasearch: iterative local improvement by dynamic programming: part I, the Traveling Salesman Problem, Technical Report, University of Twente, The Netherlands, 1995.
[41] A. Punnen, F. Gover, Ejection chains with combinatorial leverage for the TSP, Research Report, University of Colorado, Boulder, 1996.
[42] R. De Franceschi, M. Fischetti, P. Toth, A new ILP-based refinement heuristic for vehicle routing problems, Mathematical Programming, Series 105 (2006) 471–499.
[43] P. Thompson, H. Psaraftsis, Cyclic transfer algorithms for multivehicle routing and scheduling problems, Operations Research 41 (1993) 935–946.
[44] R. Ahuja, J. Goodstein, J. Liu, A. Mukherjee, J. Orlin, D. Sharma, Solving the combined through-fleet assignment model with time windows using neighborhood search, Networks 44 (2004) 160–171.
[45] R. Ahuja, A. Kumar, K. Jha, J. Orlin, Exact and heuristic algorithms for the weapon-target assignment problem, Operations Research 55 (2007) 1136–5463.
[46] R. Ahuja, J. Orlin, D. Sharma, Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem, Mathematical Programming 91 (2001) 71–97.
[47] I. Ghamlouche, T. Crainic, M. Gendreau, Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design, Operations Research 51 (2003) 655–667.
[48] M. Gendreau, T. Guertin, J. Potvin, R. Seguin, Neighborhood search heuristics for a dynamic vehicle dispatching problem with pickups and deliveries, Transportation Research, Part C 14 (2006) 157–174.
[49] K. Talluri, Swapping applications in a daily airline fleet assignment, Transportation Science 30 (1996) 237–248.
[50] F. Glover, Ejection chain moves for generalized assignment problems, manuscript, Leeds School of Business, University of Colorado, Boulder, 1997.
[51] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem, INFORMS Journal on Computing 16 (2004) 133–151.
[52] J. Rubin, A technique for the solution of massive set covering problems with application to airline crew scheduling, Transportation Science 7 (1973) 34–48.
[53] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, Proc. Principles Practice Constraint Programming, Pisa, Italy, 1998, pp. 417–431.
[54] J. Sniezek, The capacitated arc routing problem with vehicle/site dependencies: an application of arc routing and partitioning, Ph.D. Dissertation, University of Maryland, College Park, 2001.
[55] D. Espinoza, R. Garcia, M. Goycoolea, G. Nemhauser, M. Savelsbergh, Per-seat, on-demand air transportation part II; parallel local search, Transportation Science 42 (2008) 279–291.
[56] C. Archetti, M. Speranza, M. Savelsbergh, An optimization-based heuristic for the split delivery vehicle routing problem, Transportation Science 42 (2008) 22–31.
[57] R. Bent, P. Van Hentenryck, A two-stage hybrid local search for the vehicle routing problem with time windows, Transportation Science 38 (2004) 515–530.
[58] J. Linderoth, E. Lee, M. Savelsbergh, A parallel, linear programming-based heuristic for large-scale set partitioning problems, INFORMS Journal on Computing 13 (2001) 191–209.
[59] G. Cornuejols, D. Naddef, W.R. Pulleyblank, Halin graphs and the traveling salesman problem, Mathematical Programming 26 (1983) 287–294.
[60] C. Reeves, Genetic algorithms for the operations researcher, INFORMS Journal on Computing 9 (1997) 231–250.
[61] R. Ahuja, J. Orlin, Developing fitter genetic algorithms, INFORMS Journal on Computing 9 (1997) 251–253.
[62] M.O. Ball, U. Derigs, C. Hilbrand, A. Metz, Matching problems with generalized upper bound side constraints, Networks 20 (1990) 703–721.
[63] C. Aggarwal, J. Orlin, R. Tai, Optimized crossover for the independent set problem, Operations Research 45 (1997) 226–234.
[64] G. Clarke, J. Wright, Scheduling vehicles from a central depot to a number of delivery points, Operations Research 12 (1964) 568–581.
[65] K. Altinkemer, B. Gavish, Parallel savings bases heuristics for the delivery problem, Operations Research 39 (1991) 456–469.
[66] U. Derigs, A. Metz, A matching based approach for solving a delivery/pick-up vehicle routing problem with time constraints, OR-Spektrum 14 (1992) 91–106.
[67] M. Dror, M. Ball, B. Golden, Computational comparison of algorithms for inventory routing, Annals of Operations Research 4 (1986) 3–23.
[68] M. Dror, Levy, A vehicle routing improvement algorithm—comparison of a 'Greedy' and a 'Matching' implementation for inventory routing, Computers and Operations Research 13 (1986) 33–45.
[69] T.G. Crainic, M. Gendreau, J. Farvolden, A simplex-based tabu search method for capacitated network design, INFORMS Journal on Computing 12 (2000) 223–236.
[70] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, MA, 1997.
[71] G. Easwaran, H. Üster, Tabu search and benders decomposition approaches for a capacitated closed-loop supply chain network design problem, Transportation Science 43 (2009) 301–320.
[72] M. Pedersen, T. Crainic, O. Madsen, Models and tabu search metaheuristics for service network design with asset-blaance requirements, Transportation Science 43 (2009) 158–177.
[73] M. Gendreau, G. Laporte, F. Semet, Solving an ambulance location model by tabu search, Location Science 5 (1997) 75–88.
[74] A. Caprara, G. Lancia, S.K. Ng, Sorting permutations by reversals through branch-and-price, INFORMS Journal on Computing 13 (2001) 224–244.
[75] E. Johnson, M. Savelsbergh, G. Nemhauser, Progress in linear programming-based algorithms for integer programming: an exposition, INFORMS Journal on Computing 12 (2000) 2–23.
[76] G. Depuy, M.W.P. Savelsbergh, J. Ammons, L. McGinnis, An integer programming heuristic for printed circuit card assembly, Journal of Heuristics 7 (2001) 351–369.
[77] R. Marsten, Crew Planning at Delta Airlines, in: Presentation at 15th International Symposium on Mathematical Programming, Ann Arbor, MI, 1994.
[78] P.S. Ow, T.E. Morton, Filtered beam search in scheduling, International Journal of Production Rsearch 26 (1988) 279–307.

[79] M. Hifi, R. M'Hallah, T. Saadi, Algorithms for the constrained two-staged two-dimensional cutting problem, INFORMS Journal on Computing 20 (2008) 212–221.
[80] K. Hoffman, M. Padberg, Improving LP-representations of zero-one linear programs for branch-and-cut, ORSA Journal on Computing 3 (1991) 121–134.
[81] M. Savelsbergh, Preprocessing and probing techniques for mixed integer programming problems, ORSA Journal on Computing, 6, 445–454.
[82] A. Atamturk, G. Nemhauser, M. Savelsbergh, A combined lagrangian, linear programming and implication heuristic for solving large-scale set partitioning problems, Journal of Heuristics 1 (1995) 247–259.
[83] J. Kelly, J. Xu, A set partitioning heuristic for the set partitioning problem, INFORMS Journal on Computing 11 (1999) 161–172.
[84] D. Hochbaum, Approximation algoirthms for the set covering and vertex cover problem, SIAM Journal on Computing 11 (1982) 555–556.
[85] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer, New York, 1999.
[86] G. Nemhauser, L. Trotter, Vertex packing: structural properties and algorithms, Mathematical Programming 8 (1975) 232–248.
[87] W.R. Pulleyblank, Minimum node covers and 2-bicritical graphs, Mathematical Programming 17 (1979) 91–103.
[88] M.O. Ball, Locating competitive new facilities in the presence of existing facilities, in: Proceedings of the 5th Unitied States Postal Service Advanced Technology Conference, 1992, pp. 1169–1177.
[89] D. Hochbaum, Solving integer programs over monotone inequalities in three variables: a framework for half integrality and good approximations, European Journal of Operational Research 140 (2002) 291–321.
[90] M. Ball, A. Vakhutinsky, Fault tolerant virtual path layout in ATM networks, Informs Journal on Computing 13 (2001) 76–94.
[91] E. Balas, S. Seria, M. Dawande, F. Margot, G. Pataki, OCTANE: a new heuristic for pure 0–1 programs, Operations Research 49 (2001) 207–225.
[92] L.A. Hall, A.S. Shultz, D.B. Shmoys, J. Wein, Scheduling to minimize completion times: on-line and off-line algorithms, Mathematics of Operations Research 22 (1997) 513–544.
[93] M. Queyranne, Structure of a simple scheduling polyhedron, Mathematical Programming 58 (1993) 263–285.
[94] L. Wolsey, Mixed integer programming formulations for production planning and scheduling problems, Present at 12th International Symposium on Mathematical Programming, MIT, Cambridge, MA, 1985.
[95] M.X. Goemans, A supermodular relaxation of scheduling problems with release dates, in: Proceedings of 5th M P S Conference on Integer Programming and Combinatorial Optimization, in: Lecture Notes in Computer Science, vol. 1084, Springer-Verlag, 1996, pp. 288–300.
[96] M.W.P. Savelsbergh, R.N. Uma, J. Wein, An experimental study of LP-based approximation algorithms for scheduling problems, INFORMS Journal on Computing 17 (2005) 123–136.
[97] F. Raidl, An improved genetic algorithm for the multi-constrained knapsack problem, in: Proc. of the 5th IEEE Conference on Evolutionary Computation, 1998 IEEE World Congress on Computational Intelligence, Anchorage, Alaska, 1998, pp. 207–211.
[98] A. Caprara, The reversal median problem, INFORMS Journal on Computing 15 (2003) 93–113.
[99] E. Fernández, O. Meza, R. Garfinkel, M. Ortega, On the undirected rural postman problem: tight bounds based on a new formulation, Operations Research 51 (2003) 281–291.
[100] G. Nemhauser, L. Wolsey, Integer and Combinatorial Optimization-Nemhauser, John Wiley and Sons, New York, 1988.
[101] M. Fisher, The Lagrangian relaxation method for solving integer programming problems, Management Science 27 (1981) 1–18.
[102] K. Holmberg, J. Hellstrand, Solving the uncapcitated network design problem by a lagrangean heuristic and branch and bound, Operations Research 46 (1998) 247–259.
[103] P. Avello, M. Boccia, B. D'Auria, Near-optimal solution of large-scale single-machine scheduling problems, INFORMS Journal on Computing 17 (2005) 183–191.
[104] N. Brahimi, S. Dauzère-Pérès, N. Najid, Capacitated multi-item log-sizing problems with time windows, Operations Research 54 (2006) 951–967.
[105] M.F. Monaco, M. Sammarra, The beth allocation problem: a strong formulation solved by a Lagrangean approach, Transportation Science 41 (2007) 265–280.
[106] B. Thengvall, J. Bard, G. Yu, A bundle algorithm approach for the aircraft schedule recovery problem during hub closures, Transportation Science 37 (2003) 392–407.
[107] A. Caprara, M. Fischetti, P. Toth, D. Vigo, Modeling and solving the crew rostering problem, Operations Research 46 (1998) 820–830.
[108] V. Chvatal, A greedy heuristic for the set-covering problem, Mathematics of Operations Research 3 (1979) 233–235.
[109] M.L. Fisher, P. Kedia, Optimal solutions of set covering /partitioning problems using dual heuristics, Management Science 36 (1990) 674–688.
[110] A. Caprara, M. Fischetti, P. Toth, A heuristic method for the set covering problem, Operations Research 47 (1999) 730–743.