BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **11**, No 1

Sofia • 2011

Linear Integer Programming Methods and Approaches – A Survey

Krasimira Genova, Vassil Guliashki

Institute of Information and Communication Technologies, 1113 Sofia
E-mails: kgenova@iinf.bas.bg vggul@yahoo.com

Abstract: *The paper presents a survey of methods and approaches solving linear integer problems, developed during the last 50 years. These problems belong to the class of NP-hard optimization problems. To find out exact optimal solutions for this class of problems requires use of considerable computational resources. The development of efficient hybrid methods, combining in a suitable way the best features of different approaches (exact or approximate) is the actual direction, in which many researchers devote their efforts to solve successfully varioushard practical problems.*

Keywords: *Linear integer programming, exact methods, heuristic approaches and approximate algorithms.*

I. Introduction

The name linear integer programming refers to the class of combinatorial constrained optimization problems with integer variables, where the objective function is a linear function and the constraints are linear inequalities. The Linear Integer Programming (LIP) optimization problem can be stated in the following general form:

(1) Maximize $\boldsymbol{cx}$

(2) subject to: $\boldsymbol{Ax} \leq \boldsymbol{b}$,

(3) $\boldsymbol{x} \in \boldsymbol{Z}^{\boldsymbol{n}}$,

where the solution $\boldsymbol{x} \in \mathbf{Z}^{\boldsymbol{n}}$ is a vector of n integer variables: $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ and the data are rational and are given by the $m \times n$ matrix $\boldsymbol{A}$, the $1 \times n$ matrix $\boldsymbol{c}$, and the

$m \times 1$ matrix $\boldsymbol{b}$. This formulation includes also equality constraints, because each equality constraint can be represented by means of two inequality constraints like those included in (2).

A wide variety of real life problems in logistics, economics, social science and politics can be formulated as linear integer optimization problems. The combinatorial problems, like the knapsack-capital budgeting problem, warehouse location problem, travelling salesman problem, decreasing costs and machinery selection problem, network and graph problems, such as maximum flow problems, set covering problems, matching problems, weighted matching problems, spanning trees problems and many scheduling problems can also be solved as linear integer optimization problems (see, e.g. [24, 79, 104, 119]).

Some optimization problems, having nonlinear objective functions and linear constraints can be transformed in LIP optimization problems by simple approximation of the corresponding nonlinear functions by piecewise linear functions (see, e.g. [91, 116, 151]).

Solving integer programming optimization problems, that is, finding an optimal solution to such kind of problems, can be a difficult task. To solve a *nonconvex* integer programming problem could be an algorithmically unsolvable task (see, e.g. [22, 95]). For this reason such problems have not been considered in our survey. The *convex nonlinear* integer programming problems belong to the class of NP-hard problems (see, e.g. [4, 45, 116]). There does not exist an exact algorithm, which can solve these problems in time, depending polynomially on the problem input data length or on the problem size. The *linear* integer programming problems are easier solvable than the convex nonlinear integer programming problems. An instance of problem (1)-(3) can be transformed in polynomial time to an instance of a 0-1 linear integer programming problem (see, e.g. [116]). But the 0-1 linear integer programming problem can be solved by a brute-force enumerative algorithm in $O(2^n mn)$ time (see [116]). It should be noted, that there are many special cases (e.g. matching, node packing on appropriately restricted classes of graphs, and some matroid optimization problems) that belong to the class P of problems, solvable in polynomial time, i.e., there exist algorithms with polynomial time computational complexity, which can solve them. The difficulty to solve (linear and/or nonlinear) integer programming problems arises from the fact that unlike linear programming, for example, whose feasible region is a convex set, in integer programming problems, one must search for a lattice of feasible integer points to find an optimal solution. Unlike Linear Programming (LP) where, due to the convexity of the problem, we can exploit the fact that any local solution is a global optimum, the integer programming problems have many local optima and finding a *global* optimum to the problem requires one to *prove* that a particular solution dominates all the feasible points by arguments other than the calculus-based derivative approaches of convex programming with continuous variables. For this reason, the approximate algorithms solving LIP optimization problems are widely spread.

The aim of this paper is to make a survey of the techniques developed during the last 50 years to solve LIP problems and to motivate the necessity for

continuation of the researches in the area of creating more efficient metaheuristic and hybrid approaches. The paper is organized as follows: In Section II the development of exact methods for solving LIP optimization problems is considered. It is divided in three subsections as follows: *cutting planes approaches* based on polyhedral combinatorics, *enumeration techniques* and *relaxation and decomposition techniques*. Section III is devoted to some heuristics and metaheuristic approaches, as well as to population-based evolutionary algorithms, designed to solve such class of optimization problems. In Section IV some conclusions are given and directions for further research are outlined.

II. Exact methods for solving linear integer programming optimization problems

The development of exact optimization methods for LIP optimization problems during the last 50 years was very successful. There are, at least, three different approaches for solving integer programming problems, although they are frequently combined into "hybrid" solution procedures in computational practice (see [24, 80, 91, 116, 150, 151]):

- *Cutting planes* algorithms based on polyhedral combinatorics;
- Enumerative approaches and *Branch-and-Bound*, *Branch-and-Cut* and *Branch-and-Price* methods; and
- *Relaxation and decomposition* techniques.

They are considered briefly as follows below.

II. 1. Cutting Plane algorithms based on polyhedral combinatorics

The underlying idea of polyhedral combinatorics is to replace the constraint set of an integer programming problem by an alternative convexification of the feasible points and extreme rays of the problem. Both the size and the complexity of the problems solved have been increased considerably when *polyhedral theory* was applied to numerical problem solving.

In [149] W e y l proved that a convex polyhedron can alternatively be defined as the intersection of a finite number of halfspaces or as the convex hull plus the conical hull of some finite number of vectors or points. Based on this theoretical result, G o m o r y [64] derived a "cutting plane" algorithm for integer programming problems which can be viewed as a constructive proof of Weyl's theorem.

The general cutting plane approach relaxes initially the integrality restrictions on the variables and solves the resulting linear program over the constraint system (2). In case the linear program is unbounded or infeasible, the same is valid for the integer program. In case the solution to the linear program is integer, this is the optimal solution to the integer program. When the linear program has a not integer optimal solution, then a *facet-identification problem* has to be solved. Here the objective is to find a linear inequality that "cuts off" the fractional linear programming solution while assuring that all feasible integer points satisfy the inequality – that is, an inequality that "separates" the fractional point from the

polyhedron. The terminating conditions for this algorithm are as follows: 1) an integer solution is found (the LIP problem is successfully solved); 2) the linear program is infeasible and therefore the integer problem is infeasible; or 3) no cut is identified by the facet-identification procedures either because a full description of the facial structure is not known, or because the facet-identification procedures are inexact, that is, there is no possibility for algorithmically generating cuts of a known form. In case the cutting plane procedure is terminated because of the third possibility, then, in general, the search process has "tightened" the linear programming formulation so that the resulting linear programming solution value is much closer to the integer solution value.

Gomory shows in [64, 65, 66] that the fractional cutting-plane algorithm is finitely convergent with an approximate use of the lexicographic dual simplex algorithm. The proof in [117] is a reinterpretation of Gomory's proof and provides an additional insight into the nature of convergence. In [68] it is shown that given a fractional LP solution, the cuts $\Sigma_j x_j \geq 1$, where the sum is taken over all nonbasic variables cannot yield a finite fractional cutting-plane algorithm. In [20] it is proved that stronger cuts yield a finite algorithm.

A primal cutting-plane algorithm for general integer programs was proposed in [17]. A finitely convergent primal cutting-plane algorithm was proposed in [152], and simplified versions were published in [53, 153]. Because of poor computational experience, this line of research has been very inactive. An exception is a primal cutting-plane algorithm for the travelling salesman problem [119]. Although this algorithm has been moderately successful, it seems to be inferior to a fractional cutting-plane algorithm for the travelling salesman problem.

Another strategy for cutting-plane algorithms is to maintain integrality and dual feasibility and then to use cuts to obtain primal feasibility. A finite algorithm of this type has been given by G o m o r y [67]. Other similar algorithms have been proposed in [51, 52].

II. 2. Enumerative approaches

These approaches are known under different names. The most popular of them are *Branch-and-Bound*, *implicit enumeration* and *divide and conquer* (see [116]). The *explicit enumeration* is the simplest approach to solving a *pure* integer programming problem by means of enumeration of all possibilities, which are finite in number. However, due to the "combinatorial explosion" of number of these possibilities resulting from the parameter "size," only instances having relative small size could be solved by such an approach within a reasonable computational time limit. Sometimes many possibilities can be implicitly eliminated by domination or feasibility arguments. Besides straightforward or *implicit enumeration*, the most commonly used enumerative approach is called *Branch-and-Bound* (B&B), where the "branching" refers to the enumeration part of the solution technique and bounding refers to the fathoming of possible solutions by comparison to a known upper or lower bound on the solution value. The first B&B algorithm for general integer programs was introduced by L a n d and D o i g [101]. The popularity of B&B approach increased substantially after the publication of B&B

algorithm for the travelling salesman problem by Little et al. [104], because it demonstrated that large (at this time) problems could be solved by controlled enumeration. Balas [7] gave the first implicit enumeration algorithm for general 0-1 integer programming problems.

The commercial B&B codes usually relax the problem by dropping the integrality conditions and solve the resultant continuous linear programming problem over the constraint system (2). In case the solution to the relaxed linear programming problem satisfies the integrality restrictions, the solution obtained is optimal. If the linear program is infeasible, the integer program is also infeasible. Otherwise, at least one of the integer variables is fractional in the linear programming solution. Each fractional variable is chosen and a "branch" is organized, creating two subproblems which exclude the prior solution, but do not eliminate any feasible integer solutions. These new problems arising by branching of all fractional variables constitute a branching tree, and a linear programming problem is solved for each node created. Nodes can be fathomed if the solution to the subproblem is infeasible, if it satisfies all the integrality restrictions, or if it has an objective function value, which is worse than a known integer solution. A variety of strategies that have been used within the general Branch-and-Bound framework is described in [90, 91].

Early general survey articles on enumerative methods are given in [2, 13, 46, 49, 103, 110, 142, 144]. After that period many text books on integer programming have been published (see, e.g. [24, 91, 69, 116, 121, 122, 123, 138, 150, 151]).

The linear structure of LIP optimization problem (1)-(4) does not impose a strong restriction on the application of B&B algorithm. This approach can be applied also to solve Nonlinear Integer Programming (NIP) optimization problems. The greatest part of the time during the algorithm execution is spent to solve the relaxed subproblems. The choice of suitable algorithms for solving the subproblems could improve the efficiency of B&B algorithm.

II. 3. Branch-and-Cut

The bounds obtained from the LP-relaxations are often weak, which may cause standard B&B algorithms to fail in practice. It is therefore of crucial importance to tighten the formulation of the problem to be solved. The idea of dynamically adding the so called cutting planes to the problem is one way of obtaining stronger bounds. Combining the cutting plane algorithm with B&B results in the very powerful class of *Branch-and-Cut* (B&C) algorithms. The idea is to generate cutting planes throughout the B&B tree of a standard B&B algorithm, in order to get tight bounds at each node.

The B&C algorithm consists of following major components: 1) automatic reformulation procedures, 2) heuristics which provide "good" feasible integer solutions and 3) cutting plane procedures which tighten the linear programming relaxation to the linear integer problem under consideration. These components are embedded into a tree-search framework as in the B&B approach to integer programming; whenever possible, there is used a fourth component: 4) the procedure permanently fixes variables (by reduced cost implications and logical

implications) and does comparable conditional fixing throughout the search-tree. These four components are combined so as to guarantee optimality of the solution obtained at the end of the calculation. In some cases the algorithm may also be stopped prematurely to produce suboptimal solutions along with a bound on the remaining error. The cutting planes generated by the algorithm are facets of the convex hull of feasible integer solutions or good polyhedral approximations thereof and as such they are the "tightest cuts" possible. Lifting procedures assure that the cuts generated are valid throughout the search tree which aids the search process considerably and is a substantial difference to traditional (Gomory) cutting-pane approaches.

The B&C super-algorithms use all that is known about the problem. Some B&C algorithms are considered in [77, 78, 79, 120].

The increasing empirical evidence indicates that both pure and mixed integer programming problems can be solved to *proven* optimality in economically feasible computation times by methods based on the polyhedral structure of integer programs. For applications which use such Branch-and-Cut approach, see, e.g. [8, 25, 70, 107, 126, 145]. A direct outcome of these research efforts is that similar pre-processing and constraint generation procedures can be found in commercial software packages for combinatorial problems.

Various strategies for exploring the enumeration tree, together with experimental comparison, are given in [13, 15, 16, 21, 43, 47, 109]. Some theoretical results on node selection and branching strategies are presented in [44, 86, 87, 89, 134]. In [89] a family of problems is given, for which the number of nodes that must be searched for, is exponential with respect to the size of the problem, regardless of what strategies are used.

II. 4. Branch-and-Price

The philosophy of *Branch-and-Price* (B&P) is similar to the one of Branch-and-Cut. Indeed, the pricing and the cutting are procedures for tightening the LP-relaxation of the problem. In Branch-and-Price, the concept of column generation is combined with a Branch-and-Bound algorithm. The simplex algorithm arises at the origin from the column generation concept, where only variables with negative reduced costs are allowed to enter the basis at each iteration. Given a LP model with a huge number of variables, possibly depending exponentially on the instance size, it would be efficient to consider only the variables potentially improving the objective function. The main idea of column generation is to efficiently determine a variable with negative reduced costs to enter the basis, add it to the problem, resolve it and iteratively repeat this process until no variable with negative reduced costs exists anymore.

In general, the method of Dantzig-Wolfe decomposition is often used for obtaining LP/LIP models with an exponential number of variables, which provide tighter bounds than the original compact LP/LIP pair. Description of Dantzig-Wolfe decomposition is given in [116, 151].

Since column generation is an algorithm for solving LPs, it has to be combined with another method in order to solve LIPs to optimality. The B&P algorithm [9] is

the result of combining column generation with B&B. In each node of the B&B tree, column generation is performed to solve the LP-relaxation. Branching is usually performed on original variables or by other strategies to partition the remaining search space in a balanced way.

An important point is that the column generation algorithm used must be aware of branching decisions and may only generate solutions respecting them. Another interesting question is whether the column generation algorithm should search for optimal solutions of the pricing problem or not. For a detailed review of column generation and B&P methods we refer to the recent book [32]. By means of column generation the LIP problem is decomposed into a main problem and subproblems. This decomposition has natural interpretation for some combinatorial problems. Routing and scheduling are the most suitable areas for application of Branch-and-Price methods [see 33].

From a theoretical point of view, B&C and B&P are closely related, since column generation in the primal problem corresponds to cut generation in the dual and vice-versa. Furthermore, B&C and B&P can be combined in the so called branch-and-cut-and-price algorithms, where both cuts and variables are dynamically generated.

II. 5. Relaxation and Decomposition Methods

There are three wide spread approaches for relaxation of the general LIP problem, which are designed to find an upper bound of the optimal value for the maximizing LIP problem: *Linear Programming* (*LP) relaxation, Combinatorial relaxation* and *Lagrangian relaxation*. The first two approaches extend the feasible domain without any change in the objective function of the problem. The third approach provides another maximizing objective function, which has the same or greater value in a fixed feasible domain.

The *LP relaxation* for the Integer Programming model is obtained by dropping the integrality constraints on the variables. The first Branch-and-Bound algorithm using LP relaxation was described by L a n d and D o i g [101], as mentioned above. To solve the LP subproblems in the LP relaxation, a simplex-based algorithm is normally used. The adding of new rows or variables does not lead to resolving the main problem, but a re-optimization from the optimal basis of the previous step is executed. The successful developments of interior point methods for large scale linear programming [3] have attracted many researchers to direct their efforts to use the interior point methods as the LP solver in Branch-and-Bound algorithms [140].

The now commonly used variable dichotomy scheme was proposed in [31]. The treatment of general upper-bound constraints by a division scheme together with an indexing scheme was introduced in [11]. The sets considered are called specially ordered sets. This terminology is now widely used and the concept is very important in the global maximization of the piecewise linear nonconcave functions. B e a l e and F o r r e s t [12] developed this approach which enables the implementation of the division scheme without the explicit use of auxiliary integer variables.

II. 6. Combinatorial Relaxation

For realization of the combinatorial relaxation there are at least two approaches exploiting the combinatorial structure of the problem. The first approach is based on the concept of valuated matroids, introduced by Dress and Wenzel [36, 37]. Greedy-type algorithms can be used for optimization. The other approach, which is called the structural approach, utilizes algorithms to compute an upper bound on the objective function and is often based on a graph-theoretic method (see [88, 114]).

II. 7. Lagrangian Relaxation

Considering LP relaxation it was mentioned that relaxing the integrality restriction is one approach to solution of linear integer programming problems. But, this is not the only approach to relaxing the problem. The idea of dropping constraints can be embedded into a more general framework, called *Lagrangian relaxation.* This is an alternative approach, where a set of "complicating" constraints is included into the objective function in a Lagrangian fashion (with fixed multipliers that are iteratively changed). The complicating constraints are removed from the constraint set. In this way the resulting sub-problem could be solved considerably easier. The latter is necessary in order that the approach can work, because the subproblems must be repetitively solved until optimal values for the multipliers are found. The bound found by Lagrangian relaxation can be tighter than that found by Linear Programming, but only at the expense of solving subproblems in *integers*, that is, only if the subproblems do not have the *Integrality Property*. (A problem has the integrality property if the solution to the Lagrangian problem is unchanged when the integrality restriction is removed). To realize a Lagrangian relaxation it is necessary that the structure of the problem being solved is clear in order to relax then the constraints that are "complicating" (see [40]). A related approach which attempts to strengthen the bounds of Lagrangian relaxation is called *Lagrangian decomposition* (see [71]). This approach consists of isolating sets of constraints. In this way are obtained separate, easy problems to solve over each of the subsets. The dimension of the problem is increased by creating linking variables which link the subsets. All Lagrangian approaches are problem dependent. There is developed no general theory – applicable to say, in arbitrary zero-one or LIP problems.

The most Lagrangian-based strategies provide approaches which deal with special row structures. Some problems may possess a special column structure, such that when specific values are assigned to some subset of the variables, the problem is reduced to one that is easy to solve. There are decomposition algorithms, dealing with complicating variables in the problem. Benders decomposition algorithm fixes the complicating variables, and solves the resulting problem iteratively (see [14]). Based on the problem's associated dual, the algorithm must then find a cutting plane (i.e., a linear inequality) which "cuts off" the current solution point but no integer feasible points. This cut is added to the collection of inequalities and the problem is resolved.

Since each of the decomposition approaches above described provides a bound on the integer solution, they can be incorporated into a branch and bound algorithm, instead of the more commonly used linear programming relaxation. However, these algorithms are special-purpose algorithms in that they exploit the "constraint pattern" or a special structure of the problem.

As noted in [80] the computational success for difficult combinatorial optimization problems reflects the intense efforts devoted to developing the underlying polyhedral structure of these problems. Thus, in order to use this approach, one must be able to both identify specific mathematical structures inherent in the problem and then study the polyhedron associated with that structure. As more structures are understood, and can be automatically detected, we will see larger classes of problems solved by these methods. These codes will certainly be complex, but they are likely to lead to methods for solving to optimality – with reasonable computational effort – of many of the difficult combinatorial problems for which only heuristic approximate solutions are known today.

II. 8. Preprocessing

A better formulation of a LIP problem creates the possibility for its easier solution from the viewpoint of computational time and resources. All modern software systems contain modules, which apply the so called preprocessor or presolver, and which use some rules for improving the formulation of a concrete optimization problem. The basic preprocessing techniques have the aim to tighten the bounds of the variables, to fix variables, which have no influence on the optimal solution, to remove surplus constraints or to find out that the constraint system of a concrete LIP optimization problem is infeasible. The most often used techniques for preprocessing are those, described in [5, 136].

In practice some problems arise, where great input data fluctuations are available after the search process has been started. In other cases a series of related integer optimization problems has to be solved. Examples in this connection are found in many decomposition algorithms, parametrical programming algorithms, some multiobjective optimization algorithms and algorithms for analysis of mathematical models' permissibility. Sensitivity and parametric analysis of integer programs has been discussed in [29, 50, 83, 115, 135, 137, 139].

In contrast to LP optimization problems, the evaluation of the changes in the objective function's coefficients or in the right-hand side of constraint system (2) in LIP problems is more complex. Most often the investigations are connected with a concrete type of combinatorial problems.

Parallel processing presents new opportunities for computational advances in discrete optimization. An annotated bibliography and an introduction to parallelism in combinatorial optimization are given in [96, 97]. In the empirical study [127] parallel computation is simulated and it is shown that by exploring several nodes of an enumeration tree simultaneously it is possible to reduce substantially the total number of nodes that need to be considered. These results have been summarized in [128].

III. Metaheuristics and population-based evolutionary algorithms

Since the integer programming optimization problems and the LIP optimization problems (1)-(3), as mentioned above, belong to the class of NP-hard optimization problems, it is very difficult and requires great computational efforts to find out an optimal and even a feasible solution for large size problems. Very often it is more important an acceptable solution to be found out, instead to wait a long time to obtain the optimal solution. Some flexible constraints may exist in the description of the problem model and they could be changed only a little bit. The exact algorithms need to resolve the problem even in case of a little change of one constraint. This may be very time consuming and could be expensive for real applications. The approximate algorithms are not so sensitive to little changes in some constraints. Some of them solve the problem consecutively, while it is decomposed into parts. In such case the resolving of the entire problem is not necessary. The approximate algorithms as subroutines in the exact algorithms find a broad field of application. They could be used to find out a suitable initial solution or to tighten the feasible domain of solutions and to direct the search for an optimal solution. A huge number of approximate algorithms has been created for the solution of large real life LIP optimization problems without any guarantee for optimality of the final solution [141].

During the last three decades many local search based metaheuristics have been developed to avoid the trap of local optimality and to find a global optimal solution (see [130]). It was proven that they are highly useful in practice. In the last fifteen years a lot of handbooks, devoted to the basic metaheuristic approaches, to their features and characteristics, as well as to their typical applications were published (see [10, 58, 148, 106, 61, 113, 41, 48]). They are directed to scientists and operations researchers, as well as to engineers and applications specialists, who are looking for the most appropriate optimization tool to solve particular problems.

According to the quality of the solutions obtained, the approximate algorithms can be divided into three groups as follows:

- approximate algorithms having arbitrary predetermined accuracy (absolute or relative);
- approximate algorithms having in advance determined accuracy, whereat the approximation error does not tend to zero;
- heuristic algorithms – in this case it is supposed on the base of experiments and other evaluations, that with great possibility they will find out a solution of the problem with good quality using reasonable computational resources, but there is not available any guaranteed mathematical evaluation of their accuracy.

The development of approximate algorithms, for which it has been theoretically proven, that they terminate their performance using a polynomial number of standard mathematical operations, is especially important.

The basic heuristic strategies, used in the approximate algorithms could be considered as:

- constructive algorithms and
- local-improvement algorithms.

The constructive algorithms generate the solution step by step, using the data of the problem. Usually there is no solution found, until the algorithm has not terminated its performance (in contrast to the improving algorithms). To this class of algorithms belongs the so called "greedy" algorithm, where at each step a next element of the solution is included, chosen in such a way, that the best local improvement is achieved (for example the highest gain or the lowest price). One of the widely known applications of a greedy algorithm is in the travelling salesman problem [63]. These algorithms are among the fastest approximate algorithms, but they achieve often very slow quality solutions. For this reason the constructive algorithms include often some procedures of the type "look-ahead feature" [6], where the future consequences and effects from the current choice are analyzed.

The main idea of *the local-improvement algorithms* is very simple [1]. They usually start using a feasible solution of the problem, often obtained by means of a constructive algorithm. The feasible solutions in the neighbourhood $N(x)$ of the current solution x are evaluated. When any of them is better than the current solution x, it becomes the new current solution of the problem and its neighbourhood is explored. This procedure continues until there is no new improvement and the current solution at this step is a local optimum. The importance of the way for defining the neighbourhood of a solution, is evident. The neighbourhood $N(x, t)$ of solution x is the set of solutions, which can be obtained from x by means of a simple transformation t, i.e., the different transformations define different neighbourhoods. Various strategies for the choice of a new current solution x have been proposed:

- random choice of x from $N(x, t)$,
- the first solution x, for which an improvement is found, is used (first-fit),
- all solutions in the neighbourhood $N(x, t)$ are explored and the solution having the greatest improvement is chosen (best-fit),

or some other intermediate conditions are imposed.

The question about the size of this neighbourhood is important; it shows what is the distance around the current solution determining the neighbourhood for exploration of feasible solutions [108]. The big shortcoming of these algorithms is that they can guarantee only a local optimum. There are available different approaches avoiding this problem:

by means of enlargening the size of the neighbourhood $N(x, t)$ or defining different transformations t, which determine the neighbourhood $N(x, t)$,

by means of starting the search from different initial solutions, chosen randomly in the feasible domain,

through perfecting the search techniques, allowing in some cases the choice of a worse solution in the neighbourhood $N(x, t)$.

During the last 30 years some approximate algorithms have been developed of a new type, where combinations of different heuristic approaches are used. These approximate algorithms are known as *metaheuristics*. The term "metaheuristic" is introduced for the first time by G l o v e r [55] in the meaning of search for a global optimum (the highest level solution). The algorithms of this type are known also as "modern heuristics" [132]. In [147] the following definition is proposed: "The

metaheuristic is an iterative generating process, which manages subordinated heuristics through an intelligent combination of different ideas for search that explores the feasible domain, using teaching strategies for structuring the information with the aim to find an effective, near optimal solution". Each metaheuristic has one or several parameters, which have to be tuned. This makes them flexible, but for any different application (a specific class of problems), careful tuning on the base of a set of numerical examples for the problem is required. It is also necessary to make verification of the test experiments using a set of independent benchmark examples. The metaheuristics use a condition to terminate the search process. Such conditions are, for example:

- reaching in advance a given computer time or a given limit of iterations,
- expiration of a given computer time or a given iterations number, for which no improvement has been found,
- reaching a given number of changes of the current solution or a given number of evaluated solutions, and others.

The most familiar and powerful metaheuristics are *Simulated Annealing* [35, 98, 118, 125], and *Tabu Search* (see [55, 57, 59, 118, 124]). They are based on *Local Search* techniques (see [1, 10, 58, 85, 146]). Other well-known approaches in this group are Guided Local Search [146, 148], Iterated Local Search [106] and Variable Neighbourhood Search [72, 74]. The *Population-based algorithms* are a large group of metaheuristics based on the natural practices of surviving of the best that have a learning capability. These include: Genetic Algorithms (see [10, 42, 62, 81, 82, 131, 133]), Scatter Search (see [54, 56, 60]), Ant Systems/Ant Colony Optimization (see [26, 27, 28, 30, 34]), Particle Swarm Optimization (see [38, 39, 92, 93, 94, 99, 102]) and Memetic Algorithms [112, 113]. The most important heuristic approaches and algorithms are briefly considered below.

III. 1. Simulated Annealing

Simulated Annealing is one of the oldest metaheuristics, designed to avoid the local optima. It is proposed by Kirkpatrick et al. [98] in 1983 and independently by Cerny [23] in 1985. The Simulated Annealing extends the main idea of local search allowing movement towards worse solutions. The basic algorithm of Simulated Annealing is presented in [35]. In this approach the step from the solution x_k to a new solution x_{k+1} is performed with a probability:

$$P(x_{k+1} \leftarrow x_k) = e^{\left[\frac{f(x_{k+1}) - f(x_k)}{T_k}\right]} .$$

Here T_k is a parameter, called temperature. The probability for accepting the movement to x_{k+1} decreases with increase of the deterioration $\Delta = f(x_{k+1}) - f(x_k)$ of the objective function or with the decrease of the temperature T_k. The control of the possibility for accepting the new solution is realized by means of the parameter T_k, the idea for which arises from the physical annealing process. The temperature T_k is initiated usually high (i.e., the probability for movement towards a worse solution is high), and then it is gradually decreased during the ahead going search process. For

$T_k \to 0$ the behaviour of Simulated Annealing becomes the same as the one of the local search.

Different authors suggest concrete variants of this generalized algorithm of Simulated Annealing. Like each heuristic they are connected with the procedures for changing the tuneable parameters. In this case those are the temperature T_k and the criterion for terminating the search [118].

III. 2. Tabu search

This is one of the most widely used metaheuristics designed mainly to solve combinatorial optimization problems, like control of transport nets, distribution of electroenergy, schedules, etc. [61]. The main idea of this algorithm was first introduced by G l o v e r [55]. The basic algorithm includes a local search with the greatest improvement (best-fit) and a short term memory to avoid the local optima and the cycling. The short term memory is applied as a Tabu list, where the last solutions considered are stored and the movements directed towards them are forbidden. The neighbourhood of a current solution includes only solutions, which are not in the Tabu list. The set of these solutions is called allowed set. At each iteration the best solution of this set is chosen as a new current solution. This solution is included in the Tabu list and one of the solutions stored in it is removed (usually FIFO order is used). In order to avoid the local optimum, the movement towards worse neighbour solutions is allowed. Another type of memory is also used, called long term memory, where information about past search steps is stored, as well as how many times a given solution has been chosen and the frequency of changing one concrete solution, etc. This memory is used to direct the search to regions of the feasible domain, which have remained still unexplored, i.e., its purpose is to realize diversification of the search. On this occasion, Glover called this metaheuristic "Adaptive Memory Programming" (see [55, 57]). Usually the search procedure is terminated after executing a given common limit of iterations or after a given number of consecutive iterations without improving the best obtained solution.

There exist many extensions of this main idea of Tabu search (**www.tabusearch.net**). Some of them are directed to the use of an approximate objective function evaluation for the different solutions. Others use an evaluation concerning the constraint violation (the approach of Lagrangian relaxation). Different heuristics [58] for the choice of the next candidate for a solution are used, methodical and/or randomly the length of the Tabu list is changed, evaluations of the stored solutions are made with the aim of diversification of the search or making a choice of the direction to search for a new solution, etc.

III. 3. Population-based algorithms

These are metaheuristics (see [76, 143]), that unlike the previously discussed metaheuristics handle not only one, but a group, or a population of solutions. At each iteration, periods of *self-adaptation* (intensification of the search process in some region of the search space), alternate with periods of *co-operation* (information collectively gathered during the search process is used to direct further

the search). The periods of self-adaptation correspond to the execution of mutation, improvement or local search procedure, and the periods of co-operation are connected with the selection, crossover, trace updating or generation of trial points, i.e., with some (explicit or implicit) sharing among the individuals of the useful information gathered during the search. The general scheme of population-based algorithms is presented below:

Generate an initial population of individuals;
While no stopping condition is met **do**
    Co-operation,
    Self-adaptation,
EndWhile.

This approach for exploration of the feasible domain is taken from nature. The final result of these algorithms depends on the manner, in which the population is changed. Well known population-based algorithms used for discrete optimization are the evolutionary algorithms, as well as the ant colony optimization and the particle swarm optimization.

The evolutionary algorithms mimic the natural evolutionary processes [10, 62, 108, 132]. At each iteration a set of operators is applied on the individuals in the current population in order to generate the individuals of the next generation. The fitness value of each solution in the population is evaluated. To make it an objective function or another qualitative, some evaluations are used, which are obtained experimentally or in another way. The individuals, having the highest fitness are used in the next population direct or as parents generating new individuals through a change or by means of a combination between them. The operators used are: modification or mutation which changes the individuals directly, and combination or crossover between two or more individuals for generating new individuals. The remaining solutions (individuals) are rejected, i.e., a selection is performed. The evolutionary algorithms are nondeterministic algorithms. They differ from one another in the way of their presentation, evaluation, selection and change of solutions. The term evolutionary algorithms includes, besides genetic algorithms, a wide class of other population-based algorithms, since almost all functions in their common form can be free defined and adapted to the problems, for which they are applied.

Some pioneering works on *Genetic algorithms* (GA) have been published in the mid sixties (see [42, 81, 131]), but they have been further developed by H o l l a n d [82] and G o l d b e r g [62]. They use different strategies for improving the efficiency of the search (strategies for intensification of the search) [62, 125]. Their mechanism mimics the genetic evolution of species. GA use a population of feasible solutions, called *individuals*. A number of parent's pairs of individuals are selected from the current population by means of a *selection* operator. Each pair performs reproduction by means of a *crossover* operator and generates two new individuals (solutions), called an *offspring*. A *mutation* operator is used to modify randomly with a small probability the offspring individuals imitating the mutation during the natural evolution. At the end the population individuals, having worse objective function values are replaced by the corresponding better offspring

individuals. This procedure is iteratively repeated and it usually stops when the population does not improve anymore or after a fixed number of iterations (generations). Although GA have been demonstrated to work well for a variety of problems, there is no guarantee of convergence to a global optimum. Their convergence can be sensitive to the choice of genetic operators, mutation probability, selection criteria, and fine-tuning of these parameters is often required. There is a theoretical basis for the effectiveness of GA (see [62]), but in practice most problems do not fit naturally into this paradigm. It is possible genetic algorithms to be hybridized with other heuristic strategies, as well as to be organized as parallel algorithms (see, e.g. [100]).

Scatter search (SS) has been proposed by Glover [54] in 1977 to solve integer programming problems. This is a search strategy (see [55]), that generates systematically a set of *dispersed* points (solutions), from chosen *reference* points by means of convex or non-convex combinations of subsets (of two or more) reference points. The generated new points are called *trial* points. Some of them may violate the constraints in the problem. For this reason a repair procedure is used to transform the infeasible trial points into feasible points. A kind of a local search procedure is used to improve each new obtained feasible trial point. The obtained improved points form the set of dispersed points. Then a new set of reference points is selected at the next iteration among the current reference and disperse points. In comparison to GA, the reproduction in SS may be considered as multisex, because more than two parents can be matched to produce a new child.

The so called *Ant systems* (AS) have been proposed by Colorni, Dorigo and Maniezzo [26, 27, 28] in 1991. They are also population-based algorithms. AS are inspired from the behaviour of ants, searching food in the neighbourhood of their formicary using the best route to the food source. In the optimization methods of this kind, each ant is a constructive procedure that is able to generate a new solution of the problem at hand. The ants make choice by generating their corresponding new solutions on the base of two factors: the trace factor and the desirability factor [30, 34]. The first factor reflects the historical information gathered throughout the individual search of the ants. The second factor guides each ant to the choice of a solution with the best objective function value in its neighbourhood. After each iteration the ants share their new information to update the trace factor. The different AS modify the trace factor in different ways. One way, for example, is to direct the search around the neighbourhood of the best solution found so far.

Particle Swarm Optimization (PSO) has been first developed by Eberhart and Kennedy [39] in 1995 (see also [92, 93, 94]). The basic idea in PSO is to imitate the intelligent swarming behaviour, observed in flocks of birds, schools of fish, swarms of bees, etc. Each object (particle) in PSO makes steps from its current position to a new position and this motion is determined as a sum of three vectors: *inertia*, *competition* and *cooperation*. The inertia vector is determined by the current velocity of the particle $v(t)$ weighted by a constant w. In this way the tendency of the particle to maintain its current velocity is reflected. The competition vector links the current position of the particle $y(t)$ to its personal best position found during the

search process. This vector is weighted using a uniformly distributed random function. The cooperation vector links the current position of the particle $y(t)$ to the global best position, found by the particles. This vector is weighted using a second uniformly distributed random function. It is clear that the cooperation among particles is important for finding the global optimum solution. The inertia and the competition are necessary for the particle to avoid trapping in the local minima.

The term "*Memetic Algorithms*" (MA) [111] is introduced at the end of the eighties to denote a family of metaheuristics, which is a hybridization of evolutionary or any population-based approach with separate individual learning or local improvement procedures to solve a given problem. MA often apply local improvement heuristics to each individual in the population [113]. The use of local improvement heuristics directs the search procedure to regions with better solutions. Another strategy for intensification is the one, which combines the good parts of individuals. In this way the search procedure is directed to regions, where the individuals have good qualities. Techniques of this type are called "linkage learning" [75].

For the success of every metaheuristic method in solving NP-hard integer optimization problems, it is necessary to achieve both depth and breadth in the search process. There are no problems with the depth of the search in the local improvement algorithms and they can often find quickly very good solutions. In contrast to the depth, the breadth can be a critical issue for them. The population-based algorithms are better in discovering the promising areas in the variables space, since they can achieve great breadth in their search process. The exact balance between intensification and diversification in the search process is necessary for the development of efficient approximate algorithms [19]. For this reason the hybrid algorithms, combining the advantages of these two groups of algorithms are very successful [84]. Another approach for creating efficient algorithms especially designed to solve real life (combinatorial) problems is the combination of metaheuristics and logic programming with constraints [41, 105]. Some algorithms, combining metaheuristics and exact algorithms (see [130]) are developed to solve concrete classes of problems. The use of the features of a feasible domain of concrete problems leads to improvement in the quality of the solutions found.

IV. Conclusions

A large variety of different real life problems in practice are formulated as integer optimization problems. Their number and their size increase continuously. Regardless of the fact, that the productivity of exact algorithms designed to solve integer problems has been considerably improved during the last years, very often they can not be applied to solve practical problems of middle and large size because of their excessive runtimes and memory requirements. The published theoretical and also algorithmic investigations are devoted to combinatorial or binary problems. As a result, the most wide spread heuristic procedures for obtaining suitable initial solutions, evaluations of candidate-solutions, cutting planes,

specialized search strategies, etc., integrated in the commercial programming products, are effective for problems with 0-1 variables or for problems having a special structure. The solution of the integer problem in the general case remains considerably harder. The hybrid methods are promising tools, since they combine the best features of different methods (exact techniques or metaheuristics) in a complementary mode [18, 73, 84, 129, 154]. Since the obtaining of a good feasible solution in reasonable time is completely satisfactory for many practical problems, the development of heuristic algorithms, having polynomial computational complexity, is still a problem of the present day. Many large size real problems can not be solved by exact algorithms due to their exponential computational complexity. In such case the only way is the use of approximate polynomial time algorithms.

Acknowledgement: The paper is partially supported by the Bulgarian National Science Fund, Grant No DTK02/71 "Web-Based Interactive System, Supporting the Building Models and Solving Optimization and Decision Making Problems", by the European Social Fund and Bulgarian Ministry of Education, Youth and Science under Operative Program "Human Resources Development", Grant BG051PO001-3.3.04/40 and IICT-BAS research project "Modeling, Optimization and Multiple Criteria Decision Making".

References

1. Aarts, E., J. K. Lenstra. Local Search in Combinatorial Optimization. John Wiley and Sons, 1997.
2. Agin, N. Optimum Seeking with Branch-and-Bound. – Management Science, **13**, 1966, B176-B185.
3. Andersen, E. D., J. Gondzio, C. Meszaros, X. Xu. Implementation of Interior Point Methods for Large Scale Linear Programming. – In: Interior Point Methods in Mathematical Programming, T. Terlaky (Ed.), Chapter 6, Kluwer Academic Publisher, 1996, 189-252.
4. Arora, S., B. Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009.
5. Atamturk, G., L. Nemhauser, M. Savelsbergh. Conflict Graphs in Integer Programming. – European Journal of Operations Research, **121**, 2000, 40-55.
6. Atkinson, J. B. A Greedy Look-Ahead Heuristic for Combinatorial Optimization: An Application to Vehicle Scheduling with Time Windows. – Journal of Operational Research Society, **45**, 1994, 673-684.
7. Balas, E. An Additive Algorithm for Solving Linear Programs with Zero-One Variables, – Operations Research, **13**, 1965, 517-546.
8. Barahona, M., M. Grötschel, M., G. Jünger, G. Reinelt. An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design. – Operations Research, **36**, 1988, 493-513.
9. Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. – Operations Research, Vol. **46**, 1998, No 3, 316-329.
10. Bäck, T., D. B. Fogel, Z. Michalewicz. Handbook of Evolutionary Computation. New York, Oxford University Press, 1997.
11. Beale, E. M. L., J. A. Tomlin. Special Facilities in a General Mathematical Programming System for Nonconvex Problems Using Ordered Sets of Variables. – In: Proc. of the Fifth International Conference on Operational Research, J. Lawrence, (Ed.), Tavistock Publications, 1970, 447-454.

12. Beale, E. M. L., J. J. H. Forrest. Global Optimization Using Sprcial Ordered Sets. – Mathematical Programming, **10**, 1976, 52-69.
13. Beale, E. M. L. Branch and Bound Methods for Mathematical Programming Systems. – Annals of Discrete Mathematics, **5**, 1979, 201-219.
14. Benders, J. F. Partitioning Procedures for Solving Mixed-Variables Programming Problems. – Numerische Mathematik, **4**, 1962, 238-252.
15. Benichou, M., J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, O. Vincent. Experiments in Mixed Integer Linear Programming. – Mathematical Programming, **1**, 1971, 76-94.
16. Benichou, M., J. M. Gauthier, G. Hentges, G. Ribiere. The Efficient Solution of Large Scale Linear Programming Problems – Some Algorithmic Techniques and Computation Results. – Mathematical Programming, **13**, 1977, 280-322.
17. Ben-Israel, A., A. Charnes. On Some Problems of Diophantine Programming. – Cahiers du Centre d'Etudes de Recherche Operationelle, **4**, 1962, 215-280.
18. Bertacco, L., M. Fischetti, A. Lodi. A Feasibility Pump Heuristic for General Mixed-Integer Problems. – Discrete Optimization, Vol. **4**, 2007, No 1, 63-76.
19. Blum, C., A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. – ACM Computing Surveys, Vol. **35**, 2003, No 3, 268-308.
20. Bowman, V. J., G. L. Nemhauser. A Finiteness Proof for Modified Dantzig Cuts in Integer Programming. – Naval Research Logistics Quarterly, **17**, 1970, 309-313.
21. Breu, R., C. A. Burdet. Branch and Bound Experiments in Zero-One Programming. – Mathematical Programming Study, **2**, 1974, 1-50.
22. Britton, J. K. Integer Solution of Systems of Quadratic Equations. – Mathematical Proceedings of the Cambridge Philosophical Society, **86**, 1979, 385-389.
23. Cerny, V. A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulated Algorithm. – J. Optim. Theory Appl., **45**, 1985, 45-51.
24. Chen, D., R. G. Batson, Y. Dang. Applied Integer Programming: Modeling and Solution. John Wiley & Sons, 2010.
25. Chopra S., E. Gorres, M. R. Rao. Solving the Steiner Tree Problem on a Graph Using Branch and Cut. – ORSA Journal on Computing, **4,** 1992, 320-335.
26. Colorni, A., M. Dorigo, V. Maniezzo. Positive Feedback as a Search Strategy. Department of Electronics, Politechnico di Milano, Italy, Working paper, 1991, 91-16.
27. Colorni, A., M. Dorigo, V. Maniezzo. Distributed Optimization by Ant Colonies. – In: Proc. of the 1st European Conference on Artificial Life (ECAL-91), F. J. Varela and P. Bourgine (Eds.), Cambridge, MA, MIT Press, 1991, 134-142.
28. Colorni, A., M. Dorigo, V. Maniezzo. An Investigation of Some Properties of an Ant Algorithm. – In: Parallel Problem Solving from Nature, 2, R. Männer and B. Mandrieck (Eds.), Amsterdam, North-Holland, 1992, 509-520.
29. Cook, A. M., H. Gerards, A. Schrijver, E. Tardos. Sensitivity Results in Integer Programming. – Mathematical Programming, **34**, 1986, 251-264.
30. Cordon, O., F. Herrera, T. Stützle. A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. – Mathware & Soft Computing, 2002, 1-35.
31. Dakin, R. J. A Tree Search Algorithm for Mixed Integer Programming Problems. – Computer Journal, **8**, 1965, 250-255.
32. G. Desaulniers, J. Desrosiers, M. Solomon (Eds.). Column Generation. Kluwer, 2005.
33. Desaulniers G., J. Desrosiers, M. Solomon. Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. – In: Essays and Surveys in Metaheuristics, C. C. Ribeiro and P. Hansen (Eds.), Kluwer, 2001, 309-324.
34. Dorigo, M., T. Stützle. Ant Colony Optimization. Cambridge, MA, MIT Press, 2004.
35. Dowsland, K. Simulated Annealing. – In: Modern Heuristic Techniques for Combinatorial Problems, C. R. Reeves (Ed.). Blackwell, 1993, 20-69.
36. Dress, A. W. M., W. Wenzel, A New Look at the Greedy Algorithm. – Appl. Math. Lett., **3**, 1990, 33-35.
37. Dress, A. W. M., W. Wenzel. Valuated Matroids. – Adv. Math., **93**, 1992, 214-250.
38. Eberhart, R., Y. Shi. Particle Swarm Optimization: Developments, Applications and Resources. – In: Proc. Congr. Evolutionary Computation (CEC 01), Vol. **1**, 2001, 81-86.

39. E b e r h a r t, R. C., J. K e n n e d y. A New Optimizer Using Particle Swarm Theory. – In: Proc. 6th International Symposium on Micromachine and Human Science, Japan, Nagoya, 1995, 39-43.
40. F i s h e r M. L. The Lagrangian Method for Solving Integer Programming Problems. – Management Science, **27**, 1981, 1-18.
41. F o c a c c i, F., F. L a b u r t h e, A. L o d i. Local Search and Constraint Programming. – In: Handbook of Metaheuristics, F. Glover and G. Kochenberger (Eds.), International Series in Operations Research and Management Science, Vol. **57**, Kluwer Academic Publishers, 2002.
42. F o g e l, L. J., A. J. O w e n s, M. J. W a l s h. Artificial Intelligence through Simulated Evolution. New York, Wiley, 1966.
43. F o r r e s t, J. J. H., J. P. H. H i r s t, J. A. T o m l i n. Practical Solution of Large Mixed Integer Programming Problems with UMPIRE. – Management Science, **20**, 1974, 736-773.
44. F o x, B. L., J. K. L e n s t r a, A. H. G. R i n n o o y K a n n, L. E. S c h r a g e. Branching from the Largest Upper Bound: Folklore and Facts. – European Journal of Operations Research, **2**, 1978, 191-194.
45. G a r e y, M. R., D. S. J o h n s o n. Computers Intractability: A Guide to the Theory of NP-Completeness. San Francisco, W. H. Freeman, 1979.
46. G a r f i n k e l, R. S. Branch and Bound Methods for Integer Programming. – In: Combinatorial Optimization, N. Christofides, et al. (Eds.), John Wiley & Sons, 1979, 1-20.
47. G a u t h i e r, J. M., G. R i b i e r e. Experiments in Mixed-Integer Programming Using Pseudo-Costs. – Mathematical Programming, **12**, 1977, 26-47.
48. M. Gendreau, J. Y. Potvin (Eds.). Handbook of Metaheuristics. Series "International Series in Operations Researches/Management Science". Vol. **146**. 2nd Ed. Springer, 2010.
49. G e o f f r i o n, A. M., R. E. M a r s t e n. Integer Programming Algorithms: A Framework and State-of-the-Art Survey. – Management Science, **18**, 1972, 465-491.
50. G e o f r i o n, A. M., R. N a u s s. Parametric and Postoptimality Analysis in Integer Linear Programming. – Management Science, **18**, 1977, 453-466.
51. G l o v e r, F. A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem. – Operations Research, **13**, 1965, 879-919.
52. G l o v e r, F. A Pseudo Primal-Dual Integer Programming Algorithm. – Journal of Research of the National Bureau of Standards, **71B**, 1967, 187-195.
53. G l o v e r, F. A New Foundation for a Simplified Primal Integer Programming Algorithm. – Operations Research, **23**, 1968, 434-451.
54. G l o v e r, F. Heuristics for Integer Programming Using Surrogate Constraints. – Decision Sciences, **8**, 1977, 156-166.
55. G l o v e r, F. Future Paths for Integer Programming and Links to Artificial Intelligence. – Computers and Operations Research, Vol. **13**, 1986, No 5, 533-549.
56. G l o v e r, F. Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms). – Discrete Applied Mathematics "Viewpoints on Optimization", 1994, No 49, 231-255.
57. G l o v e r, F. Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges. – In: Advances in Metaheuristics, Optimization and Stochastic Modeling Technologies, Barr, Helgason and Kennington (Eds.), Boston, MA., Kluwer, 1997, 1-75.
58. F. Glover, G. Kochenberger (Eds.). Handbook of Metaheuristics. Vol. **57.** International Series in Operations Research & Management Science. Norwell, MA, Kluwer Academic Publishers, 2003.
59. G l o v e r, F., M. L a g u n a. Tabu Search. Kluwer Academic Publishers, 1997.
60. G l o v e r, F., M. L a g u n a, R. M a r t. Fundamentals of Scatter Search and Path Relinking. – Control and Cybernetics, Vol. **39**, 2000, No 3, 653-684.
61. G l o v e r, F., M. L a g u n a. Tabu Search. – In: Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.). Oxford University Press, 2002, 194-208.
62. G o l d b e r g, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Mass., Addison Wesley, 1989.
63. G o l d e n, B., L. B o d i n, T. D o y l e, J. W. S t e w a r t. Approximate Traveling Salesman Algorithm. – Operations Research, **28**, 1980, 694-711.

64. G o m o r y, R. E. Outline of an Algorithm for Integer Solution to Linear Programs. – Bulletin American Mathematical Society, **64**, 1958, 275-278.
65. G o m o r y, R. E. Solving Linear Programming Problems in Integers. – In: Combinatorial Analysis, R. E. Bellman and M. Hall, Jr. (Eds.), American Mathematical Society, 1960, 211-216.
66. G o m o r y, R. E. An Algorithm for Integer Solutions to Linear Programs. – In: Recent Advances in Mathematical Programming, R. Graves and P. Wolfe (Eds.), McGraw-Hill, 1963, 269-302.
67. G o m o r y, R. E. An All-Integer Programming Algorithm, in Industrial Scheduling. J. F. Muth and G. I. Thompson (Eds.), Prentice-Hall, 1963, 193-206.
68. G o m o r y, R. E., A. J. H o f f m a n. On the Convergence of an Integer-Programming Process. – Naval Research Logistics Quarterly, **10**, 1963, 121-123.
69. G r ö t s c h e l, M., L. L o v a s z, A. S c h r i j v e r. Geometric Algorithms and Combinatorial Optimization. Berlin, Springer, 1988.
70. G r ö t s c h e l, M., C. L. M o n m a, M. S t o e r. Computational Results with a Cutting Plane Algorithm for Designing Communication Networks with Low-Connectivity Constraint. Report No 187, Schwerpunktprogramm der Deutschen Forschungsgemeinschaft, Universität Augsburg, 1989.
71. G u i g n a r d, M., S. K i m. Lagrangian Decomposition: A Model Yielding Stronger Lagrangian Bounds. – Mathematical Programming, **39**, 1987, 215-228.
72. H a n s e n, P., N. M l a d e n o v i ć. An Introduction to Variable Neighborhood Search in Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, S. Voß, S. Martello, I. Osman, and C. Roucairol (Eds.), Kluwer Academic Publishers, 1999, 433-438.
73. H a n s e n, P., N. M l a d e n o v i c, D. U r o š e v i c. Variable Neighborhood Search and Local Branching. – Computers & Operations Research, Vol. **33**, 2006, No 10, 3034-3045.
74. H a n s e n, P., N. M l a d e n o v i ć, J. A. M o r e n o P e r e z. Variable Neighborhood Search: Methods and Applications. – Annals of Operations Research, Vol. **175**, 2010, 367-407.
75. H a r i k, G. Linkage Learning via Probabilistic Modeling in the ECGA. Tech. Rept. IlliGal 99010, University of Illinois, Urbana-Champaign, 1999.
76. H e r t z, A., D. K o b l e r. A Framework for Description of Population Based Methods. – In: Tutorials and Research Reviews, 16th European Conference on Operational Research Brussels, Belgium, 1998, 48-59.
77. H o f f m a n K. L., M. P a d b e r g. LP-Based Combinatorial Problem Solving. – Annals Operations Research, **4,** 1985, 145-194.
78. H o f f m a n, K. L., M. P a d b e r g. Improving the LP-Representation of Zero-One Linear Programs for Branch-and-Cut. – ORSA Journal Computing, **3**, 1991, 121-134.
79. H o f f m a n, K. L., M. P a d b e r g. Solving Airline Crew Scheduling Problems by Branch-and-Cut. – Management Science, **39**, 1993, 657-682.
80. H o f f m a n, K. L., M. P a d b e r g. Combinatorial and Integer Optimization. – Solution Techniques for Integer Programming, 1999.
http://www.esi2.us.es/~mbilbao/combiopt.htm#combi4
81. H o l l a n d, J. H. Outline for a Logic Theory of Adaptive Systems. – Journal of the ACM, Vol. **9**, 1962, Issue 3, 297.
82. H o l l a n d, J. H. Adaptation in Natural and Artificial Systems. Ann Arbor, MI, The University of Michigan Press, 1975.
83. H o l m, S., D. K l e i n. Three Methods for postoptimal Analysis in Integer Linear Programming. – Mathematical Programming Study, **21**, 1984, 97-109.
84. H o p p e r, E. Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods. PhD Thesis, Cardiff University, UK. 2000.
85. H o o s, H., T. S t ü t z l e. Stochastic Local Search – Foundations and Applications. San Francisco, Morgan Kaufmann, CA, 2004.
86. I b a r a k i, T. Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms. – Journal of Computer and Information Science, **5**, 1976, 315-344.
87. I b a r a k i, T. Power of Dominance Relations in Branch-and-Bound Algorithms. – Journal of the Association for Computing Machinery, **22**, 1977, 463-468.
88. I w a t a, S., K. M u r o t a. Combinatorial Relaxation Algorithm for Mixed Polynomial Matrices. – Math. Program., Ser. A 90, 2001, 353-371.

89. J e r o s l o w, R. G. Trivial Integer Programs Unsolvable by Branch-and-Bound. – Mathematical Programming, **6**, 1974, 105-109.
90. J o h n s o n, E. L., S. P o w e l l. Integer Programming Codes. Design and Implementation of Optimization Software, H. J. Greenberg (Ed.), NATO Advanced Study Institute Series, Sijthoff & Noordhoff, 1978, 225-248.
91. M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. L. Pulleyblank, G. Reinelt, G. Rinaldi, L. A. Wolsey (Eds.). 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, Springer, 2009.
92. K e n n e d y, J., R. C. E b e r h a r t. Particle Swarm Optimization. – In: Proc. of IEEE International Conference on Neural Networks, N. J. Piscataway (Ed.), 1995, 1942-1948.
93. K e n n e d y, J., R. C. E b e r h a r t. A Discrete Binary Version of the Particle Swarm Algorithm. – In: Proc. IEEE Int. Conf. Systems, Man, and Cybernetics: Computational Cybernetics and Simulation, Vol. **5**, 1997, 4104-4108.
94. K e n n e d y, J., R. C. E b e r h a r t, Y. S h i. Swarm Intelligence. San Francisco, Morgan Kaufmann Publishers, 2001.
95. K h a c h i y a n, L. Convexity and Algorithmic Complexity Solving Polynomial Programming Problems. – Technitcheskaya Kibernetika, **6**, 1982, 47-56 (in Russian).
96. K i n d e r v a t e r, G. A. P., J. K. L e n s t r a. Parallel Algorithms. – In: O'hEigeartaigh et al. (Eds.), 1985, 106-128.
97. K i n d e r v a t e r, G. A. P., J. K. L e n s t r a. An Introduction to Parallelism in Combinatorial Optimization. – Discrete Applied Mathematics, **14**, 1986, 135-156.
98. K i r k p a t r i c k, S., C. G e l l a t, M. V e c c h i. Optimization by Simulated Annealing. – Science, **220**, 1983, 671-680.
99. K r u s i e n s k i, D. J., W. K. J e n k i n s. Design and Performance of Adaptive Systems, Based on Structured Stochastic Optimization Strategies. – IEEE Circuits and Systems, Vol. **5**, First Quarter 2005, No 1, 8-20.
100. K u, M. -Y., M. H. H u, M. -J. W a n g. Simulated Annealing Based Parallel Genetic Algorithm for Facility Layout Problem. – International Journal of Production Research, 2010. **http://www.informaworld.com/smpp/content~db=all~content=a921530351~tab=content**
101. L a n d, A. H., A. G. D o i g. An Automatic Method for Solving Discrete Programming Problems. – Econometrica, **28**, 1960, 97-520.
102. L a s k a r i, E. C., K. E. P a r s o p o u l o s, M. N. V r a h a t i s. Particle Swarm Optimization for Integer Programming, 2002. **http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.4972&rep=rep1&type=pdf**
103. L a w l e r, E. L., D. E. W o o d. Branch-and-Bound Methods: A Survey. – Operations Research, **14**, 1966, 699-719.
104. L i t t l e, J. D. C., K. G. M u r t y, D. W. S w e e n e y, C. K a r e l. An Algorithm for the Traveling Salesman Problem. – Operations Research, **11**, 1963, 972-989.
105. L i t t l e, J. Integer Programming, Constraint Logic Programming and Their Collaboration in Solving Discrete Optimisation Problems. PhD Thesis, Department of Mathematical Sciences, Brunel University, 2000.
106. L o u r e n c o, H. R., O. M a r t i n, T. S t ü t z l e. Iterated Local Search. – In: Handbook of Metaheuristics, F. Glover and G. Kochenberger (Eds.). – International Series in Operations Research and Management Science, Vol. **57**, Kluwer Academic Publishers, 2002, 321-353.
107. M a g n a n t i, T. L., R. V a c h a n i. A Strong Cutting Plane Algorithm for Production Scheduling with Changeover Costs. – Operations Research, **38,** 1990, 456-473.
108. M i c h a l e w i c z, Z., D. B. F o g e l. How to Solve It: Modern Heuristics. New York, Springer-Verlag, 2000.
109. M i t r a, G. Investigations of some Branch and Bound Strategies for the Solution of Mixed Integer Linear Programs. – Mathematical Programming, **4**, 1973, 155-170.
110. M i t t e n, L. G. Branch-and-Bound Methods: General Formulation and Properties. – Operations Research, **18**, 1970, 24-34.
111. M o s c a t o, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

112. M o s c a t o, P. Memetic Algorithms: A Short Introduction. – In: New Ideas in Optimization, D. Corne et al. (Eds.), McGraw Hill, 1999, 219-234.
113. M o s c a t o, P. Memetic Algorithms. – In: Handbook of Applied Optimization, P. Pardalos and M. Resende (Eds.), Oxford University Press, 2002, 157-167.
114. M u r o t a, K. Combinatorial Relaxation Algorithm for the Maximum Degree of Subdeterminants: Computing Smith-McMillan Form at Infinity and Structural Indices in Kronecker Form. – Appl. Algebra Engin. Comm. Comput., **6**, 1995, 251-273.
115. N a u s s, R. M. Parametric Integer Programming. Columbia, University of Missouri Press, 1979.
116. N e m h a u s e r, G. L., L. A. W o l s e y. Integer and Combinatorial Optimization. New York, Chichester, Brisbane, Toronto, Singapore, John Wiley & Sons, 1988.
117. N o u r i e, F. J., E. R. V e n t a. An Upper Bound on the Number of Cuts Needed in Gomory's Method for Integer Forms. – Operations Research Letters, **1**, 1982, 129-133.
118. O s m a n, H., G. L a p o r t e. Meta-Heuristics: A Bibliography. – Annals of Operations Research, **63**, 1996, 513-628.
119. M. W. Padberg, Ed. Combinatorial Optimization. – Mathematical Programming Study, **12**, 1980.
120. P a d b e r g, M., G. R i n a l d i. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. – SIAM Review, **33**, 1991, 60-100.
121. P a d b e r g, M. Linear Optimization and Extensions. Heidelberg, Springer-Verlag, 1995.
122. P a d b e r g, M., M. P. R i j a l. Location Scheduling and Design in Integer Programming. Kluwer's Academic Publishers, MA, 1996.
123. P a r k e r, R. G., R. L. R a r d i n. Discrete Optimization. San Diego, Academic Press, 1988.
124. P e d r o s o, J. P. Tabu Search for Mixed Integer Programming. Technical Report Series: DCC-2004-02, 2004.
http://www.dcc.fc.up.pt/~jpp/publications/PDF/mip-ts-WP.pdf
125. P i r l o t, M. Heuristic Search Methods. – In: Oper. Res. Designing Practical Solutions; Tutorial and Research Review Papers, Euro XIII/OR 36, The Joint EURO/Oper. Res. Society Conference; University of Strathclyde, Glasgow, 19-22 July, 1994, 180-201.
126. P o c h e t, Y., L. A. Wolsey. Solving Multi-Item Lot Sizing Problems Using Strong Cutting Planes. – Management Science, **37**, 1991, 53-67.
127. P r u u l, E. Parallel Processing and Branch-and-Bound Algorithm. M. S. Thesis, Cornell University, 1975.
128. P r u u l, E., G. L. N e m h a u s e r, R. R u s h m e i e r. Parallel Processing and Branch-and-Bound: A Historical Note. – Operations Research Letters, Vol. **7**, 1988, Issue 2, 65-69.
129. P u c h i n g e r, J., G. R. R a i d l. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification, IWINAC 2005. – In: Lecture Notes in Computer Science. Berlin, Springer, 2005, 41-53.
130. P u c h i n g e r, J. Combining Metaheuristics and Integer Programming for Solving Cutting and Packing Problems. PhD Thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms. Supervised by G. R. Raidl and U. Pferschy, January 2006.
131. R e c h e n b e r g, I. Cybernetic Solution Path of an Experimental Problem, Royal Aircraft Establishment Transl., 1122, B. F. Toms Transl. Ministry of Aviation, Royal Aircraft Establishment, Farnborough, Hants, United Kingdom, 1965.
132. R e e v e s, C. R. Modern Heuristic Techniques for Combinatorial Problems. McGraw-Hill, 1995.
133. R e i d, D. J. Enhanced Genetic Operators for the Resolution of Discrete Constrained Optimization Problems. – Computers & Operations Research, Vol. **24**, 1997, No 5, 399-411.
134. R i n n o o y K a n n, A. H. G. On Mitten's Axioms for Branch and Bound. – Operations Research, **24**, 1976, 1176-1178.
135. R i n o o y K a n, A. H. G. An Introduction to the Analysis of Approximation Algorithms. – Discrete Applied Mathematics, **14**, 1986, 111-134.
136. S a v e l s b e r g h, M. W. P. Preprocessing and Probing for Mixed Integer Programming Problems**. –** ORSA J. on Computing, **6**, 1994, 445-454.
137. S c h r a g e, L., L. A. W o l s e y. Sensitivity Analysis for Branch and Bound Integer Programming. – Operations Research, **33**, 1985, 1008-1023.
138. S c h r i j v e r, A. Linear and Integer Programming. New York, Wiley, 1984.
139. S h a p i r o, J. F. Sensitivity Analysis in Integer Programming. – Annals of Discrete Mathematics, **1**, 1977, 467-477.

140. Da Silva, A., D. Abramson. A Parallel Interior Point Method and Its Application to Facility Location Problems. – Computational Optimization and Applications, Vol. **9**, 1998, Issue 3, 249-273.
141. Silver, E. A. An Overview of Heuristic Solution Methods. – Journal of the Operational Research Society, **55**, 2004, 936-956.
142. Spielberg, K. Enumerative Methods in Integer Programming. – Annals of Discrete Mathematics, **5**, North Holland, 1979, 139-183.
143. Taillard, E. D., L. M. Gambardella, M. Gendreau, J.-Y. Potvin. Adaptive Memory Programming: A Unified View of Metaheuristics, Tutorials and Research Reviews. – In: 16th European Conference on Operational Research Brussels, Belgium, 1998, 30-38.
144. Tomlin, J. A. Branch-and-Bound Methods for Integer and Non-Convex Programming. – In: Integer and Nonlinear Programming. J. Abadie, Ed., American Elsevier, 1970, 437-450.
145. VanRoy, T. J., L. A. Wolsey. Solving Mixed Integer Programming Problems Using Automatic Reformulation. – Operations Research, **35**, 1987, 45-57.
146. Verachi, S., S. Prestwich. Constructive Vs. Perturbative Local Search for General Integer Linear Programming. – In: 18th Irish Conference on Artificial Intelligence and Cognitive Science, 2007.
http://sysrun.haifa.il.ibm.com/hrl/lscs2008/papers/03_constructive.pdf
147. Voss, S. Metaheuristics: The State of the Art. – In: Local Search for Planning and Scheduling, Lecture Notes in Computer Science, A. Nareyek (Ed.), Springer, 2001, 1-23.
148. Voudouris, C., E. Tsang. Guided Local Search, Handbook of Metaheuristics. F. Glover and G. Kochenberger (Eds.). International Series in Operations Research and Management Science, Kluwer Academic Publishers, 2002.
149. Weyl, H. Elementare Theorie der konvexen Polyheder. – Comm. Math. Helv., **7**, 1935, p. 290, (Translated in Contributions to the Theory of Games, **1**, 1950, No 3).
150. Williams, H. P. Logic and Integer Programming. – In: International Series in Operations Research & Management Science. Springer, 2009.
151. Wolsey, L. Integer Programming. – In: Wiley – Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., 1998.
152. Young, R. D. A Primal (All Integer), Integer Programming Algorithm. – Journal of Research of the National Bureau of Standards, **69B**, 1965, 213-250.
153. Young, R. D. A Simplified Primal (All Integer), Integer Programming Algorithm. – Operations Research, **16**, 1968, 750-782.
154. Yuh-Chyun, L., M. Guignard, Chun-Hung Chen. A Hybrid Approach for Integer Programming Combining Genetic Algorithm, Linear Programming and Ordinal Optimization. – Journal of Intelligent Manufacturing, **12**, 2001, 509-519.