We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

172,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Simulation Needs Efficient Algorithms

Marian Chudy
Military University of Technology
Poland

1. Introduction

Many simulation systems which can be applied, for example, to solve problems in manufacturing, supply chain management, financial management and war games need procedures that have to make current decisions in limited time. It means that the procedures, mentioned above, should work efficiently and give the right or almost right solutions at the right time. The time the procedures need to prepare a solution depends on the quality of an algorithm and the “size” of the problem the algorithm solve. It leads us to the computation theory. To omit very theoretical considerations we assume that:

- the size of a problem denotes the length of the chain of its input data,
- the time complexity of an algorithm denotes the number of elementary operations it takes to solve the problem in the worst case,
- the time complexity of a problem denotes the complexity of the best possible, may be unknown yet, algorithm.

The chain of input data should be unambiguous and expressed in the shortest possible way. It is important, because the time the procedure needs to put the chain of data into a computer must be the shortest one. The more precisely definitions base on the idea of Turing machine. It is necessary to mention that there are other kinds of complexity like complexity in average sense or best behavior.

The theory of computation bases on decision problems for uniformity. The formal definition of the decision problem we can find for example in (Hromkovic, 2001). Without any losses of generality we define the decision problem as a problem which requires only a “yes” or “not” answer; some authors call such problem as a recognition problem.

There are many problems which are decision problems inherently. The Satisfiability Problem (SAT) is one of paradigmatic decision problems. It is to decide, for a given formula in the Conjunction Normal Form (CNF), whether it is satisfiable (output: “yes” = “1”) or not (output: “no” = “0”).

Decision problems are strongly connected with optimization problems which rely on finding the “best” solution among all solutions in some set of the feasible solutions. An objective function and the goal (min, max) determines which of the feasible solution is the best. The formal definition of an optimization problem we can find, for example, in (Hromkovic, 2001).

For every optimization problem there exists a set of decision problems which are connected with it.

Moreover, we can indicate a method that solves the optimization problem by solving the series of the decision problems from the set we mention above. The example given below describes the relation between optimization and decision problem.

The 0-1 optimization problem is to find

$$x^* \in S \subset B^n = \{x \in E^n : x_j \in \{0,1\} \text{ } j = \overline{1,n}\}$$

such that

$$(c | x^*) = \min_{x \in S} (c | x) = \min_{x \in S} \sum_{j=1}^n c_j x_j \quad (1)$$

where

$$S = \left\{ x \in B^n : \sum_{j=1}^n a_{ij} x_j \leq d_i \quad i = \overline{1,m} \right\} \quad (2)$$

The set of the decision problems relevant to the optimization problem (1), (2) is to decide whether exists vector $x \in B^n$ such that

$$(c | x) = \sum_{j=1}^n c_j x_j \leq y, \quad x \in S \quad (3)$$

for given integers y and S defined by (2). The answer is “yes” or “no”.

We observe that taking some sequence of integers y and solving relevant to them problems (3), one can obtain an optimal solution of the optimization problem (1), (2).

To refer to the previous definitions it is worth to add that

- the set of all instances of the optimization problem (1), (2) is determined by the parameters n, c, A, d .
- the set of all instances of the relevant decision problem (3) is determined by parameters n, c, A, d, y .

2. Measuring the efficiency of algorithms

The introduction given below justifies our focus on the complexity as a main measure of the algorithm efficiency.

To describe the characteristics of an algorithm time complexity more precisely one should introduce a few necessary objects and definitions.

Let us denote by

$N = \{0,1,2,\dots\}$ the set of all natural numbers,

R_+ - the set of non negative real numbers

and let $f : N \rightarrow R_+$ be a function and F be the set of such functions.

Definition 1 Let $g : N \rightarrow R_+$ be a function. We define

$$O(g(n)) = \{f \in F : \exists a \geq 0, n_0 \in N \text{ such that } \forall n \in N, n \geq n_0 f(n) \leq a \cdot g(n)\}.$$

If $f(n) \in O(g(n))$ we say that the function f does not grow asymptotically faster than the function g .

Definition 2 Let $g : N \rightarrow R_+$ be a function. We define

$$\Omega(g(n)) = \{f \in F : \exists b \geq 0, n_1 \in N \text{ such that } \forall n \in N, n \geq n_1 f(n) \geq b \cdot g(n)\}.$$

If $f(n) \in \Omega(g(n))$ we say that the function f grows asymptotically at least as fast as the function g .

If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ we say that f and g are asymptotically equivalent. It means that

$$\begin{aligned} f(n) \in \Theta(g(n)) &= O(g(n)) \wedge \Omega(g(n)) = \\ &= \left\{ t \in F : \exists c \geq 0, d \geq 0, m \in N \text{ such that } \forall n \in N, n \geq m \right. \\ &\quad \left. t(n) \leq c \cdot g(n), \quad t(n) \geq d \cdot g(n) \right\}. \end{aligned}$$

Let us denote by $f_\alpha(n)$ the complexity of an algorithm α , n - the size of a problem.

Definition 3 We say that an algorithm α is polynomial time one when

$f_\alpha(n) \in O(p(n))$, where $p(n)$ some polynomial for $n \in N$. It means that

$$\exists c \geq 0, n_0 \in N \text{ such that } \forall n \in N, n \geq n_0 f_\alpha(n) \leq c \cdot p(n),$$

in other cases the algorithm α is called exponential time algorithm.

The definition 3 allows us to describe the fundamental classes of decision problems.

The class P (polynomial) consists of all these decision problems for which a polynomial time algorithm exists, maybe unknown yet.

The class NP (non deterministic polynomial) can be define in many different but equivalent ways. It seems that more useful definition which allows to verify if a decision problem belongs to the class NP is the following one.

The class NP consists of all these decision problems for which the “yes” answer is verifiable by some procedure it takes polynomial time.

The class P is a subclass of NP ($P \subset NP$). It comes from the definition of P and NP ; every instance of P with answer “yes” is verifiable in polynomial time because it is obtained in polynomial time from definition of P . One can explore the special subclasses of NP and also search the classes of problems which contain NP class.

Let us denote by

D_Π - the set of all instances of the decision problem Π ,

$x(z)$ - the input chain for $z \in D_\Pi$; the symbols of the chain $x(z)$ belongs to some

defined alphabet Σ ,
 $N(z)$ - the size of instance $z \in D_{\Pi}$ (length of the chain $x(z)$).
There is only one $x(z)$ for given $z \in D_{\Pi}$.

Definition 4 We say that problem Π_1 transforms to problem Π_2 in polynomial time, $\Pi_1 \propto \Pi_2$, when there exists a polynomial time algorithm (function) α such that for every $z \in D_{\Pi}$ and $x(z)$, $\alpha(x(z)) = y(s)$, $s \in D_{\Pi_2}$ where $y(s)$ - the input chain of some instance $s \in D_{\Pi_2}$.

It means that for every chain $x(z)$, we can compute in polynomial time adequate chain $\alpha(x(z)) = y(s)$ which is the input chain of some instance $s \in D_{\Pi_2}$.

The polynomial transformation is a useful tool to investigate unknown problems basing on information about related known problems.

Definition 5 A problem Π is called *NP – hard* if, for every $\Pi' \in NP$, $\Pi' \propto \Pi$.

A problem Π is called *NP – complete* if, $\Pi \in NP$ and Π is *NP – hard*.

The following comes from the definition above: an optimization problem does not belong to *NP* class, but when a related decision problem belongs to *NP – complete* class, we call the optimization problem *NP – hard*.

From the definition of *NP – complete* class results that if, someone found polynomial time algorithm for solving a *NP – complete* problem, then all problems from *NP* class would be solved in polynomial time. This conclusion comes from the definition of *NP – complete* class and polynomial transformation.

Summarizing our considerations we can say that efficient algorithm means algorithm with complexity bounded by some polynomial. The following table confirms this thesis by some important relations.

Time complexity function	Size of largest instances solvable in 1 hour	
	With present computer	With computer 100 times faster
n	N_1	$100N_1$
n^2	N_2	$10N_2$
n^3	N_3	$4,63N_3$
2^n	N_4	$N_4 + 6,64$
3^n	N_5	$N_5 + 4,19$

Table 1. Competition between efficiency of algorithms and computers

We should add that any complexity results for the decision problem also hold for the original(optimization) problem i.e. the optimization problem is not much harder than related decision problem.

To classify the efficiency of an algorithm more precisely some remarks about $\Theta(g(n))$ notation and the little-0-notation is necessary.

Property 1 The following relation holds

$$f(n) \in \Theta(g(n)) \text{ if and only if } g(n) \in \Theta(f(n)).$$

The set $\Theta(g(n))$ contains these functions which are equivalent to function $g(n)$.

The set $\Theta(g(n))$ is called the category of complexity which is represented by function $g(n)$.

Property 2 All logarithmic functions belong to one category $\Theta(\log n)$.

It means that if, $a > 1, b > 1$ then $\log_a n \in \Theta(\log_b n)$.

Definition 6 We say that function $f(n)$ belongs to $o(g(n))$ if

$$\forall c > 0 \exists N(c) \in N \forall n \geq N(c) f(n) \leq c \cdot g(n).$$

Intuitively, it means that function $g(n)$ grows much faster than function $f(n)$.

Property 3 Exponential functions do not belong to one category of complexity i.e.

$$\text{if } 0 < a < b \text{ then } a^n \in o(b^n)$$

i.e. if $0 < a < b$ then $\forall c > 0 \exists N(c) \in N \forall n \geq N(c) a^n \leq c \cdot b^n$.

Now, we introduce nine categories of complexity:

$$\begin{aligned} c(1) &= \Theta(\log_2 n), c(2) = \Theta(n), c(3) = \Theta(n \log_2 n), c(4) = \Theta(n^2), \\ c(5) &= \Theta(n^i), c(6) = \Theta(n^j), c(7) = \Theta(a^n), c(8) = \Theta(b^n), c(9) = \Theta(n!) \\ \text{where } j &> i > 2 \text{ and } b > a > 1. \end{aligned}$$

Property 4 If $c(r) = \Theta(g(n))$ and $f(n) \in c(s)$ for $s < r$ then

$$f(n) \in o(g(n)).$$

It means: the lower category the better complexity.

2. Examples of the exact efficient algorithms

The set of all possible algorithms that one can use to solve a problem we divide into two parts: exact algorithms and approximation algorithms. Precise definitions of these two kind of algorithms one can find in (Hromkovic, 2001). Roughly speaking, an approximation algorithm gives a solution that is closed the optimal solution. At this part we focus our attention on exact algorithms. At the previous part of the chapter we establish the difference between the complexity of a problem and the complexity of an algorithm it solves.

From the definition, the complexity of a problem cannot be worse than the complexity of an algorithm the problem solves.

In the majority we do not know the complexity of a problem. Improving the algorithms they solve the problem we bring nearer to this unknown value.

2.1 Advanced sorting algorithms

Sorting problem is one of the most important in computer science and practice, including simulation. A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order or lexicographical one. Many algorithms, such as search and merge algorithms require sorted lists to work correctly and efficiently. So, the sorting algorithms that prepare the data to them should be also efficient. We assume that the size of sorting problem is the number of elements at list to be sorted. The piece of data actually used to determine the sorted order is called the key. Most of the algorithms in use

have the complexity that belongs to $O(n^2)$ category or $O(n \log n)$; the lower bound for them is $O(n)$. It is difficult to establish only one criterion for judging sorting algorithms because many algorithms that have the same complexity do not have the same speed on the same input. Despite of this we assume that the complexity is the main criterion of the efficiency. A second (additional) criterion for judging sorting algorithm is their space (memory) requirement. Do they require extra space or can the list be sorted in place without additional memory beyond a few variables? A third (additional) criterion is stability. An algorithm is stable if it preserves the order of keys with equal values. Taking into account only the main criterion the class of advanced sorting algorithms contains these of them which have $O(n \log n)$ complexity, for example merge, quick sorts and heap. Unfortunately, the very known algorithms like bubble, insertion, selection and shell sorts belong to the class $O(n^2)$.

Quick sort (Partition Exchange Sort) algorithm is a divide and conquer algorithm which relies on a partition. To partition a list, we choose an element, called a pivot. Each round of quick sort contains the following actions:

- move all elements smaller than pivot before the pivot,
- move all elements greater than pivot after the pivot,
- recursively quick sort the values (sublist) before the pivot,
- recursively quick sort the values (sublist) after the pivot.

The most complex issue in quick sort is choosing a good pivot. Bad choices of pivot can result in drastically slower $O(n^2)$ performance. If at each step we choose the median as the pivot then quick sort works in $O(n \log n)$. Quick sort is instable and needs constant space.

Merge sort is also a divide and conquer technique. It has two phases: divide phase and conquer phase. The divide phase repeatedly divides the list into smaller sublists until the sublist is small enough to sort. The conquer phase relies on sorting simpler sublists and merging to the overall list. To realize this idea we should know: how to sort sublist (a list

containing smaller number of values) and how to merge two sorted lists. The complexity of this algorithm is $O(n \log n)$. Merge sort is stable but needs additional space.

Heap sort is an efficient version of selection sort. This algorithm works in following way:

- determine the largest (smallest) element of the list and place that at the end (or beginning) of the list,
- continue this operation with the rest of the list.

These tasks are accomplished efficiently by using a data structure called a heap which is special type of binary tree. Heap sort is stable, runs in $O(n \log n)$ time and needs constant space.

Parallel computation is the most promising method to obtain real speed up in sorting. The ideal model is Parallel Random Access Machine (PRAM). To find the largest element of the table $A[1..n]$ we need $n(n-1)/2$ processors p_{ij} $i, j = \overline{1, n}$, $1 \leq i \leq j \leq n$

and additional table $T[1..n]$ with all elements $T[i] = 1, i = \overline{1, n}$ at the start moment.

We assume that processors can simultaneously read and write the same part of memory (Concurrent Read, Concurrent Write memory access). Each processor p_{ij} compares elements $A[i], A[j]$ and writes in table $T[1..n]$ the following values $T[i] = 0$ if $A[i] < A[j]$, $T[j] = 0$ if $A[i] > A[j]$. After all comparisons we obtain one element $T[k] = 1$ and number k denote the index of largest element from table $A[1..n]$. The complexity of this procedure is equal to $O(1)$. Repeating this procedure for the series of reduced tables we obtain the ordered one. This computing model is not realistic but indicates a direction of thinking (Neapolitan & Naimipour, 2004).

2.2 Polynomial time algorithm for solving linear programming problem

Linear programming (LP) is one of the most frequently used optimization method. It is used to prepare decision in the industrial, economic and military activity. It is also important that linear programming is often used as a part of more complex models in optimization, for example as a relaxed model in discrete optimization. The linear programming problem (LPP) relies on minimization (or maximization) of linear function subject to linear constraints. One of the form of the linear programming problem is

$$\max(c | x) = \max \sum_{j=1}^n c_j x_j \quad (4)$$

$$\text{subject to} \quad Ax \leq d \quad (5)$$

$$x \geq 0 \quad (6)$$

where $A = (a_{ij})_{m \times n}$, $x \in E^n$, $c \in E^n$, $d \in E^m$.

This problem is called primal problem.

For each primal problem exists dual problem. The dual problem related to (4) – (6) is given below

$$\min(d \mid y) = \min \sum_{i=1}^m d_i y_i \quad (7)$$

$$\text{subject to} \quad A^T y \geq c \quad (8)$$

$$y \geq 0 \quad (9)$$

where $y \in E^m$.

There are many relations between primal (4) – (6) and dual (7) – (9) problems.

They are often applied in constructing method for solving LPP.

The most important are:

- the primal and dual LPP either have the optimal solutions or they do not have ones,
- for optimal solutions x^* and y^* the following relation holds $(c \mid x^*) = (d \mid y^*)$,
- if for some feasible solutions $x, y, (c \mid x) = (d \mid y)$ holds then $x = x^*, y = y^*$.

The simplex method and its many modifications are the most useful and most important methods for solving LPP yet. The essence of this method relies on searching adjacent vertices of polyhedral (5), (6) to find the vertex which is the optimal solution. Algebraic equivalent of this is to search adjacent bases and nonnegative base solutions of the linear equation system $Ax = d$. This conception results from the following theorem.

Theorem 1 If there exists an optimal solution x^* of linear programming problem (4) – (6) then there exists a vertex $\bar{x}$ of the polyhedral (5), (6) such that $(c \mid x^*) = (c \mid \bar{x})$.

It allows one to search only the vertices of the polyhedral defined by (5), (6) because the linear programming problem belongs to the class of convex problems and it means that every local minimum is equal to the global minimum of LPP. As it was shown by (Klee & Minty, 1972), the simplex method has the exponential complexity in the worst case. It is in contradiction with the practical experiences. This problem was explained by (Shamir, 1987). He obtained, for simplex method, a quadratic lower bound on the average number of pivots (steps from one basic to an adjacent one). This outcome confirms the good quality of the simplex method.

There are at least two methods which are better than the simplex method, taking into account their complexity. Older of them is the ellipsoid method developed by (Khachiyan, 1979), which is the first polynomial algorithm for solving linear programming problem. The exact description of the ellipsoid algorithm would take large space. We will present only short explanation of this algorithm. In fact the ellipsoid algorithm computes the sequence of feasible solutions of the system of strong inequalities related to LPP (4) – (6). We assume that

data of the problem is integer. For given system of inequalities $Ax \leq d$ the size of the

$$\text{problem is } L = \sum_{i=1}^m \sum_{j=0}^n \lceil \log_2(|a_{ij}| + 1) + 1 \rceil, \quad a_{i0} = d_i, i = \overline{1, m}$$

The system $Ax \leq d$ has a solution if and only if the system

$$(a_i | x) < d_i + 2^{-L}, i = \overline{1, m} \quad (10)$$

has a solution.

The set of feasible solutions of the system (10) should be nonempty.

We start to estimate the maximum value of the objective function $(c | x)$ with checking the feasibility of the following system: $(c | x) \geq \alpha_0, Ax \leq d, x \geq 0$. (11)

If the set of feasible solutions is nonempty, we know that the optimum value is lesser than α_0 . We may now decrease α_0 by factor 2 and check for feasibility again. If this is true, we

know that $(c | x^*) \in [\alpha_0 / 2, \alpha_0)$. We get the optimum in a number of steps polynomial in the input size, each step being a call to feasibility checking polynomial algorithm. The feasibility checking algorithm for current value α is the main part of the ellipsoid algorithm. It relies on computing sequences of ellipsoids $E_k(x^k), x^k$ - the

centre of this ellipsoid, such that
$$\frac{\text{vol}E_{k+1}(x^{k+1})}{\text{vol}E_k(x^k)} < 1.$$

So, the volume of ellipsoid $E_{k+1}(x^{k+1})$ is substantially smaller than the volume of the previous ellipsoid. This is the most important point in this method because the polynomial complexity follows this property. The computing of this algorithm ends when the centre of an ellipsoid is the feasible solution of the system (11) for current value α or the set of feasible solution is empty.

It is necessary to add that the ellipsoid method can also start with the following system of inequalities: $Ax \leq d, x \geq 0, A^T y \geq c, y \geq 0, (c | x) \geq (d | y)$.

The iteration complexity of the ellipsoid method is $O(n^2 L)$ but computational complexity in worst case is $O(n^4 L)$.

In 1984 Narendra Karmarkar introduced a new idea for solving linear programming problem.

Let us consider the following pair of linear programming problems:

- primal problem
$$\min_{x \in R_0} (c \mid x) = \min_{x \in R_0} \sum_{j=1}^n c_j x_j \quad (12)$$

where
$$R_0 = \{x \in E^n : Ax = d, x \geq 0\}$$

- dual problem
$$\max_{y \in Q_0} (d \mid y) = \max_{y \in Q_0} \sum_{i=1}^m d_i y_i \quad (13)$$

where
$$Q_0 = \{y \in E^m : A^T y \leq c\}$$

We assume that
$$R_0^+ = \{x \in E^n : Ax = d, x > 0\} \neq \emptyset \text{ and}$$

$$Q_0^+ = \{y \in E^m : A^T y < 0\} \neq \emptyset$$

Denoting by $w = c - A^T y$, $X = xI^n$, I^n - diagonal matrix
the Karush-Kuhn-Tucker (KKT) conditions are

$$Ax = d, x \geq 0, \quad A^T y + w = c, w \geq 0, \quad Xw = 0 = (0, 0, \dots, 0) \quad (14)$$

For each pair (x, y) which satisfies conditions (14) x is the optimal solution of primal problem (12) and y is the optimal solution of dual problem (13).

The relaxed form of KKT conditions is

$$Ax = d, x \geq 0, \quad A^T y + w = c, w \geq 0, \quad Xw = 1\mu = (\mu, \mu, \dots, \mu) \quad (15)$$

We obtain (14) from (15) when $\mu = 0$.

Following (Sierksma, 1996) we have.

Theorem 2 Let $R_0 = \{x \in E^n : Ax = d, x \geq 0\}$ be bounded. Then (15) has for each positive μ ($\mu > 0$) a unique solution, denoted by $x(\mu), y(\mu), w(\mu)$.

Definition 7 The sets $\{x(\mu) : \mu > 0\}, \{y(\mu) : \mu > 0\}$ are called the interior path of problem (12) and (13) respectively. For shortness, we will call $x(\mu)$ the interior path of primal problem and $y(\mu)$ the interior path of dual problem. The parameter μ is called interior path parameter.

Theorem 3 For each interior path parameter μ , the duality gap satisfies

$$(c \mid x(\mu)) - (d \mid y(\mu)) = n\mu \quad (16)$$

Basing on these properties we can form the general steps of the algorithm. Let x^k be the current interior point of the feasible region (x^k is not the point of interior path but only

corresponds to the point $x(\mu^k)$, and μ^k be the current interior path parameter. The algorithm determines next interior point x^{k+1} which is closer to interior path than x^k . This new point is given by the following expression

$$x^{k+1} = x^k + s(x^k, \mu^k)$$

where $s(x^k, \mu^k)$ is the search direction which causes x^{k+1} to be closer the interior path than x^k . Next, the algorithm decreases the interior parameter according to formula below

$$\mu^{k+1} = \eta \mu^k, \quad \eta \in (0,1)$$

The procedure is repeated for pair (x^{k+1}, μ^{k+1}) , until a pair (x^+, μ^+) has been reached for which the stop criteria $n\mu^+ \leq \varepsilon$ is satisfied. Since $\mu^+ \approx 0$, and $n\mu^+$ is approximately equal to the duality gap $(c | x^+) - (d | y^+)$ then vectors x^+ and y^+ approximate optimal solutions of primal and dual problems respectively; $x^+ \approx x^*$, $y^+ \approx y^*$. We omit the explanation of this part of algorithm that deal with the search direction $s(x^k, \mu^k)$. This needs large space to describe it. We also omit the problem of searching start point and the prove of polynomiality of the interior path method. The complexity of the first interior algorithm (Karmarkar, 1984) is:

- iteration complexity - $O(nL)$,

- computation complexity - $O(n^3 L)$, where L denotes the length of input chain. The Karmarkar's idea has been improved and extended. Now, there are many new interior point methods with better complexity. For example, an infeasible-interior-point algorithm (Anstreicher et al., 1999), applied to random linear programs generated according to a model of Todd gives an exact solution in expected number of iterations $O(n \log n)$.

2.3 Efficient algorithms in graph theory

The graph theory is one of the most important tools for modeling and solving problems. The area of its applications is very broad. To refer to the title of the chapter we will give only short presentation dealing with selected but important problems and algorithms. An efficient algorithm can be obtained using greedy approach. A greedy algorithm iteratively makes one greedy choice (local optimum) after another, reducing each given problem into a smaller one, and never reconsider its choices. A suitable structure of the problem guarantees the efficiency of the greedy algorithm. For many other problems (with improper structure), greedy algorithms fail to produce the optimal solutions. They sometime produce the unique

worst possible solutions. Despite this, greedy algorithm is often used because it is faster than other methods in many cases.

The minimum spanning tree (MST) belongs to the class of problems that have suitable structure to use greedy approach. This problem is formulated as follows:

for given connected graph $G = (V, E)$ with weighted edges, find the tree $T = (V, F)$, $F \subset E$, such that the graph T is connected and the sum of the weights of edges in T is as small as possible.

Telephone companies are interested in minimum spanning tree, because the minimum spanning tree of set sites determine the wiring scheme that connects the sites using minimum wire. The greedy algorithms of Prim and Kruskal solve this problem efficiently (Neapolitan & Naimipour, 2004).

Prim's algorithm continuously increases the size of a tree starting with a single arbitrary vertex until it spans all the vertices:

1. take an arbitrary vertex $v_1 \in V$, $V^T = \{v_1\}$, $F = \emptyset$
2. chose edge $(v_1, v^*) \in E$ such that $w(v_1, v^*) = \min_{(v_1, v) \in E} w(v_1, v)$, (greedy step)
 where $w(v_1, v)$ denotes the weight of edge (v_1, v) ,
3. set $F = \{(v_1, v^*)\}$ and $V^T = \{v_1, v^*\}$
4. repeat steps 2 and 3 for actual values F and V^T until $V^T = V$.

The complexity of this algorithm is $O(|V|^2) = O(n^2)$.

Kruskal's algorithm starts with a graph $T = (V, F)$, $F = \emptyset$ and the ordered set E_o containing all edges from E in the order of increasing weights. We assume that each vertex is connected component and each component is a tree.

For $k = 1, k \leq n - 1$ with step equals one do: select the next smallest weight edge (greedy step) and if the edge connects two different connected components then add the edge to F . The complexity of this algorithm is $O(n^2 \log n)$.

One has observed that for sparse graph Kruskal's algorithm is better than Prim's but for dense graph Prim's algorithm is better than Kruskal's one.

The single-source shortest path problem arises in transportation and communications. For given connected graph $G = (V, E)$ with weighted edges, which is a model of structure of a transportation or communications system, the important problem is to find the shortest paths from given vertex (source) to a destination or to all the others. This problem is related to the spanning tree problem because the graph representing all the paths from given vertex to all the others must be a spanning tree. Dijkstra's algorithm (greedy approach) solves this problem in polynomial time.

The running time of this algorithm is estimated by expression $O(n^2)$.

Greedy approach can be effectively used to solve the problem of data compression. This problem relies on finding the minimum length bit string which can be used to encode a string of symbols. For example, the problem of text compression is: what is the smallest number of bits one can use to store a piece of text. The Huffman's algorithm (greedy approach) generates optimal code of symbols by related binary tree. The complexity of the Huffman's algorithm is $O(n \log n)$, where n denotes the number of different symbols in a file.

3. Approaches to solving NP-hard problems

The common opinion is that the majority of practical problems belong to the $NP - hard$ class. It means, from the definition that for any such problem does not exist a polynomial algorithm that solves it. This obliges us to search special approaches to these problems that give acceptable, from the practical point of view, results. We will present only a few such approaches.

Denote by D_{Π} the set of all instances of given $NP - hard$ problem Π .

The first approach relies on exploring the set D_{Π} to find some subset $D_{\Pi_s} \subset D_{\Pi}$ of "easy" instances and designing an efficient algorithm for solving special problem Π_s for which the set of instances is equal to D_{Π_s} . To explain this approach we will consider $NP - hard$ problem Π , which is the linear integer programming problem (LIPP):

$$(c | x^*) = \max_{x \in S} (c | x) = \max_{x \in S} \sum_{j=1}^n c_j x_j \quad (17)$$

$$\text{where } S = \{x \in E^n : Ax \leq d, x \geq 0, x - \text{int.}\} \quad (18)$$

and the relaxation of the problem (17), (18), which is the linear programming problem (LP):

$$(c | x^o) = \max_{x \in T} (c | x) = \max_{x \in T} \sum_{j=1}^n c_j x_j \quad (19)$$

$$\text{where } T = \{x \in E^n : Ax \leq d, x \geq 0\} \quad (20)$$

The special problem Π_s can be obtained by selecting special properties of constraint matrix A or special properties of the objective function $(c | x)$.

An excellent example for the first case is integer linear programming problem with totally unimodular matrix A .

Definition 8 An integer matrix A is totally unimodular (TUM) if every square submatrix of A has determinant either 0, 1, or -1.

Theorem 4 Let A be totally unimodular matrix. For each integer vector d the polyhedron (20) has only integer vertices. The proof one can find in (Sierksma, 1996).

Theorem 5 If A is totally unimodular and d is an integer vector, then optimal solution x^o of LP is also optimal solution of ILPP i.e. $x^o = x^*$. The proof results from theorem 1, 4 and relation $S \subset T$.

So, we can solve special integer linear programming problem (17), (18) with totally unimodular matrix A by solving its polynomial relaxation (19), (20).

Another special problem Π'_s , one can obtain from general problem (17), (18) taking into account an original property of the objective function $(c \mid x)$.

The sequence (c_j) is called superincreasing

$$\text{when} \quad \sum_{i=1}^{j-1} c_i < c_j \quad \text{for} \quad j=2,3,\dots \quad (21)$$

We will consider sequences containing n elements only and assume that for $n=1$ a sequence is a superincreasing one.

Theorem 6 If the problem (17)-(18) satisfies the following assumptions:

- a sequence (c_j) is the superincreasing and non negative one,
- elements a_{ij} are non negative ($a_{ij} \geq 0$),

then the optimal solution of the problem (17)-(18) is given by the following procedure

$$x_j^* = \begin{cases} 1 & \text{when } a_j \leq d - \sum_{k \in N_j^+} a_k \\ 0 & \text{otherwise} \end{cases} \quad j = n, n-1, \dots, 1 \quad (22)$$

where

a_j - the j -th column of the constraint matrix A ,

$$d = (d_1, d_2, \dots, d_m)^T, \quad N_n^+ = \emptyset$$

$$N_j^+ = \left\{ k : x_k^* = 1, \quad k \in \{n, n-1, \dots, j+1\} \right\}.$$

The proof results from (21) and assumptions.

The complexity of the procedure (22) is equal to $O(n^3)$, (Chudy, 2007).

Theorem 6 allows us to solve special case of optimization problem (17), (18) in polynomial time, when the assumptions it needs are satisfied.

The second approach relies on designing exponential algorithm which belongs to the lower category than the algorithms known at present. Promising results base on new concept of complexity which is called parameterized complexity. The theory of parameterized complexity was developed by Rod Downey and Michael Fellows. An introduction to the new field was presented in the monograph (Downey & Fellows, 1999). This is two-dimensional complexity theory where complexity is measured in terms of input size and some parameter of a problem. The concept of the parameterized complexity is motivated by the observation that there exist several hard problems that require exponential time of computing when the complexity is measured in terms of the input size only, but they are computable in polynomial time in the input size and exponential in a selected one parameter of the problem. It worth to note (Hromkovic, 2001) that the concept of parameterized complexity includes the concept of pseudo-polynomial complexity.

The interesting version of parameterized algorithm we find in (Reed et al, 2004). For given graph $G = (V, E)$ with m edges and n vertices the algorithm settles either a set $V_1 \subset V$ of at most k vertices which intersects every odd cycle, or the information that no

such set exists. The running time is $O(4^k kmn)$.

The third approach bases on designing an approximation algorithm that gives reasonable feasible solution of the given problem. The formal definition of the approximation algorithm and its properties is presented in (Hromkovic, 2001). The book by Vazirani, (Vazirani, 2003) contains many precisely selected problems and approximation algorithms that provide solutions whose quality are good enough.

We will consider the problem (17), (18) and try to find an approximation algorithm using the superincreasing sequence (21) renumbering, if necessary, all variables of this problem and assume that the sequence (c_j) is nonnegative and none decreasing. To obtain upper-bound of optimal objective function value, we will introduce new objects (Chudy, 2007).

Let us denote by

H^n - set of all finite superincreasing integer sequences (h_j) , $j = \overline{1, n}$,

$A^n = \{h \in H : h_j \geq c_j, j = \overline{1, n}\}$ - the set of finite superincreasing sequences with integer elements no smaller than suitable elements of the sequence (c_j) .

Remembering that (c_j) is non decreasing we form the following definition.

Definition 9 A superincreasing sequence $h^* = (h_j^*)$ is called the nearest up to the

sequence (c_j) when $h^* \in A^n$, $\|c - h^*\| = \min_{h \in A^n} \|c - h\| = \min_{h \in A^n} \sum_{j=1}^n |c_j - h_j|$.

The complexity of the procedure that compute this sequence is equal to $O(n^2)$.

The upper-bound

$$\sum_{j=1}^n h_j^* x_j \geq \sum_{j=1}^n c_j x_j^* \quad (23)$$

of optimal objective function value for the problem (17), (18) is given by (23)

where $x = (x_j)$, $j = \overline{1, n}$ denotes a feasible solution computed by procedure (22)

when we set the sequence (h_j^*) instead of the sequence (c_j) in (17), (18) and

$x^* = (x_j^*)$, $j = \overline{1, n}$ denotes an optimal solution of the problem (17), (18),

under assumption $a_{ij} \geq 0$, $c_j \geq 0$. The assessment (23) we obtain in polynomial time.

The presented approaches can be combine to obtain more efficient method that give optimal or almost optimal solution of the hard problem. It is necessary to say that we have omitted in our considerations such important methods like randomized algorithms, evolutionary algorithms, quantum computing and parallel computing.

4. Conclusion

We must repeat that high quality of simulation system needs efficient algorithms. The algorithms we have described above deal with only part of the areas we are interested in. The presented short review indicates that almost all problems including hard ones can be solved efficiently enough. The designer of a simulation system should posses a set of tools that support him in selecting proper methods which could satisfy the requirements.

5. References

- Anstreicher, K.; Ji, J. & Potra, F. (1999). Probabilistic Analysis of An Infeasible-Interior-Point Algorithm for Linear Programming. *Mathematics of Operations Research*, Vol.24, No.1, (January 1999), page numbers (176-192), ISSN 0364-765X
- Chudy, M. (2007). Remarks on 0-1 optimization problems with superincreasing and superdecreasing objective functions, BOOK OF ABSTRACTS 23rd IFIP TC7 Conference on System Modelling and Optimization, pp.392-393, ISBN978-83-88309-0, Cracow, Poland, July 2007, University of Science and Technology, Cracow
- Downey, R. & Fellows, M. (1999) *Parameterized Complexity*, Springer-Verlag, ISBN 978-0-387-94883-6, Berlin
- Hromkovic, J. (2001). *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Springer-Verlag, ISBN 3-540-66860-8, Berlin
- Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, Vol.4, No. 4, (1984), page numbers (373-395), ISSN 0209-9683
- Khachiyan, L. (1979). A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, vol. 20, No.1, (1979), page numbers (191-194), ISSN 0197-6788

- Klee, V. & Minty, G. (1972). How good is the simplex algorithm, In: *Inequalities, III*, Sisha, O. page numbers (159-175), Academic Press, ISBN 0125641141, New York
- Neapolitan, R. & Naimipour, K. (2004). *Foundations of Algorithms. Using C++ Pseudocode* Jones and Bartlett Publishers, Inc, ISBN 07637-23878, Boston
- Reed, B.; Smith, K. & Vetta, A. (2004) Finding Odd Cycle Transversals. *Operations Research Letters*, Vol. 32, No. 4, July 2004, page numbers (299-301), ISSN 0167-6377
- Shamir, R. (1987). The Efficiency of the Simplex Method: A Survey. *Management Science*, Vol. 33, No. 3, (March 1987), page numbers (301-334), ISSN 0025-1909
- Sierksma, G. (1996). *LINEAR AND INTEGER PROGRAMMING Theory and Practice*, Marcel Dekker, Inc. ISBN 0-8247-9695-0, New York
- Vazirani, V., V. (2003). *Approximation Algorithms*, Springer-Verlag, ISBN 3-540-65367-8, Berlin

IntechOpen

IntechOpen

IntechOpen

Modeling Simulation and Optimization - Focus on Applications

Edited by Shkelzen Cakaj

ISBN 978-953-307-055-1

Hard cover, 312 pages

Publisher InTech

Published online 01, March, 2010

Published in print edition March, 2010

The book presents a collection of chapters dealing with a wide selection of topics concerning different applications of modeling. It includes modeling, simulation and optimization applications in the areas of medical care systems, genetics, business, ethics and linguistics, applying very sophisticated methods. Algorithms, 3-D modeling, virtual reality, multi objective optimization, finite element methods, multi agent model simulation, system dynamics simulation, hierarchical Petri Net model and two level formalism modeling are tools and methods employed in these papers.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Marian Chudy (2010). Simulation Needs Efficient Algorithms, Modeling Simulation and Optimization - Focus on Applications, Shkelzen Cakaj (Ed.), ISBN: 978-953-307-055-1, InTech, Available from:

<http://www.intechopen.com/books/modeling-simulation-and-optimization-focus-on-applications/simulation-needs-efficient-algorithms>

INTECH
open science | open minds

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](https://creativecommons.org/licenses/by-nc-sa/3.0/), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen