Optimization of Gas Transport – Evolutionary Calculations on a Small Cluster of Computers

Igor Mračka¹, Tibor Žáčik¹, Marek Hyčko¹, Rudolf Hajossy¹, Peter Somora²

¹ Mathematical Institute, Slovak Academy of Sciences, Štefánikova 49, SK–814 73 Bratislava, Slovakia
² Altova GmbH, Rudolfplatz 13a/9, A-1010 Wien, Austria

ABSTRACT

The purpose of the paper is to demonstrate the experience with distributed computing systems using evolutionary strategies for optimization problems in the gas industry. Two different approaches of distributed computing (cooperating agents and Client-Server) are presented.

INTRODUCTION

Transient optimization of a gas transport is a long-term challenge [2], [5]. Various optimization procedures have been proposed in the past, which excessively simplified the transport system: e.g., eliminated technological constraints on compressors and pipelines. In addition, a network without controllers or check valves is often assumed. However, these simplifications can lead to inapplicable solutions.

The aim of this article is to optimize the transition between two steady states using a sufficiently realistic simulation, with a complete optimization lasting no more than 30 min. It means to get sufficiently accurate results in a time sufficiently short for practice.

Our design is based on an already developed gas transport simulator (for 1 CPU) that takes into account a large number of real technological and contract constraints [8], [9]. The optimization solution based on existing simulator opens a possibility of using other simulators in a similar way.

Increasing the realism of the model can complicate the shape of the optimized function (multimodality, constraints, discontinuity in the working area), so that efficient methods of linear, nonlinear or dynamic programming cannot be used. Therefore, we focused on stochastic population optimization algorithms. Our earlier experience of using such methods in steady state optimization [6] directed us to utilize it also for transient tasks.

Such an algorithm like Evolution Strategies (ES) [1] needs to quantify the value of an objective function, while the function itself can be multidimensional, nonlinear, non-differential, multimodal, steep and shallow. On the other hand, these algorithms (evolutionary algorithms) require a much greater number of function evaluations and there is no guarantee that the global optimum will be found at a given time. For real world tasks where position of global optimum is unknown, it is not necessary to find the global extreme, but the main effort is to find the best possible solution up to a given time.

Our task is therefore to find such a real valued vector (characterizing the control of compressors during the whole simulation of transition between two steady states) that fulfilling all specific technological and contract constraints and the total gas consumption or time of transition is minimal.

We started with improving ES optimization of steady state [6] by using easy to implement cooperating agent strategies (Island model in [7]). In this concept each computer works separately with its own small population and computers exchange best results among each other using UDP broadcast. Later the method was extended to solve the transient problem.

To master transient optimization task in a meaningful time, it is almost necessary to use all available parallelization on multicore CPUs and as well to distribute the calculations among the largest number of available computers (all available computers in our experimental conditions and possible other remote machines).

After experiments with the Island model, the Client-Server method based on TCP/IP was developed. (The latter method is harder to implement than the former one.) The Client-Server
method enables significantly extend the size of a computer cluster.

**Parallelization on Local Network Clusters of PCs**

We have combined four concepts to achieve our goal.

1. Robust stochastic evolutionary optimization methods that require a large amount of calculations of an objective function.
2. Available gas transport simulator long time developed and optimized for a single-core CPU.
3. Ability to implement simulations as independent processes on a multi-core CPU.
4. Ability to communicate with computers on the network and use their collective computing power.

This approach has following benefits:
- Possibility to solve optimization tasks that were previously impossible due to computational complexity.
- The ability to work with a debugged long-term used simulator that has been for historical reasons progressively developed and fully optimized for single-core CPUs while taking full advantage of modern multi-core hardware.
- The use of installed power on the local network increases the independence on external computing resources. (So, even in the case of an outbound Internet communication failure, the required calculations can still be performed.)
- Distributed networking architecture operating on the local network is also standardly supported by external computing providers, such as Google Cloud, Amazon Web Services, etc., so the developed solution can be also extended by using remote services.

**Evolution Strategies**

Population based stochastic optimization algorithms are methods that use population of individuals, where each member contains parameters of an objective (optimized) function as well as additional information that are used to control behavior of an individual during generating an offspring. For each individual the value of an objective function is evaluated. Then these values are used to select a population of parents. One or more parents then try to generate one or more new offspring with respect to their additional information. It leads to larger generation of offspring, which is evaluated again and similarly as before the new generation of parents is selected and the whole process repeats. One of the examples of such methods is Evolution Strategies (ES). For more information see [1] (simple block diagram of algorithm is depicted in Fig. 1).

![Block diagram of Evolution Strategies algorithm](image)

**Fig. 1 Block diagram of Evolution Strategies algorithm. Between selection and mutation the crossover operation could be placed, but we do not use it in our experiments. Stop condition in tested cases was 30 min time limit.**

**steady state optimization**

Our earlier experience with gas transport network optimization by ES dealt with steady state optimization [6].

The task for a stationary optimization is determined as follows:

We have a model of gas network, which consists of pipelines, valves, pressure regulators, check valves and compressor stations, along with filters, compressors and coolers (see illustrative Fig. 14). States of all valves (which are open or closed) determine the network topology. For the input and output nodes, the required pressures or flow rates are determined. Examples of such tasks are calculation of the new daily regime, and calculation of initial and final states in transient optimization task.

For available compressors, it is necessary to determine which of them will be connected and to find their revolutions which define the stationary state of the pipeline network in which all technological and contract constraints are fulfilled.

In [6] authors showed that use of a suitable stationary calculation, the number of sought parameters can be reduced. For each compressor station, instead of the numbers of compressors and their revolutions only a single parameter – the compression ratio of this compressor station – will be sought. It is sufficient to assign the compression ratio for each compressor station of the network, and the stationary calculation will find all pressures and flows across the entire network, and, with
built-in optimization, the best number of compressors and their revolutions. A compression ratio equal to one is also permitted, which means that this compressor station is in a by-pass state.

Such a reduction of the dimensionality of the problem is not necessary, but if used, it will greatly accelerate the calculation and will raise the accuracy of the stationary optimization. Further acceleration of calculations can be achieved by parallelization of individual simulations on multi-core computers. The rate of calculation increases in proportion to the number of cores available.

We realized that the stochastic nature of the method does not always find a solution that is close enough to the optimal one. It is possible that, once in a while, the method will not find the by-pass state of a compressor station although it is much more convenient. Therefore, if multiple computers are available, it is advisable to increase the computing power and to move to a distributed calculation on a cluster of computers.

**COOPERATING AGENTS**

First approach is similar to Island model in [7]. Assume that we have multiple computers that are connected to the same subnet so that they are visible to each other using the User Datagram Protocol (UDP see [3]). For initial implementation, we have selected this protocol to communicate because it is simple. Its "unreliability" that, in theory, some packets of information may not be delivered, is unimportant to a certain degree. On individual computers, an optimization calculation is performed and it represents an agent which is capable of self-operation (see Fig. 2).

An agent – an optimization program – is modified to receive and broadcast "broadcast messages" on a commonly defined port. A protocol is defined by which the program can use a script language to send and receive commands for running scripts, such as, e.g., to make settings of the calculation by evolutionary strategies and to give commands to start and end the optimization calculation. E.g., using script:

```plaintext
BroadcastMessage("RUN SCRIPT
task(1).setup.setNumberOfParents (20)")
BroadcastMessage("RUN SCRIPT
task(1).setup.setNumberOfOffsprings (40)")
BroadcastMessage("RUN SCRIPT
task(1).setup.setNumberOfGenerations (50)")
BroadcastMessage("START TASK 1")
```

It sends commands to run scripts which set the numbers of parents and descendants and the number of required generations for evolutionary strategies to all computers on local subnet. The last is the command to start the calculation itself. Of course, the calculations will only run on computers on which the appropriate optimization calculation is installed and running and broadcasting is enabled on the appropriate port.

It follows that there will be one control computer that will trigger the calculations and should be able to obtain results from other agents. Indeed, using UDP, optimization calculations regularly sent to other computers their best results. Thus each optimization program has knowledge of the best results currently obtained by other calculations. It would be suitable to apply this information during its own calculation.

![Fig. 2 The scheme of a broadcast domain with independent Agents 1, 2, …, n, communicating among themselves via UDP/IP protocol. Each agent runs a separate ES-algorithm with a unique population broadcasting the best found individuals to other agents.](image)

The results communicated by each other include information on optimized parameters – compression ratios, their respective sigma parameters and the values of fitness function. They are sent when the calculation finds a result better than the best till now. On the other hand, received individuals from other calculations are included in a new generation of parents found by a calculation itself and after the arrangement according to the fitness function the number of individuals of the generation of parents will be reduced back to the desired number of parents (see Figs. 3 and 1).

Thus, the best results of other calculations become a natural part of their own calculation and a synergic effect arises. Due to a stochastic nature, computations are searching different areas of the space parameters on different computers, so that the search is more complex and therefore more accurate and thanks to parallelism even faster.
Thanks to the method used, the control computer permanently keeps the best results from all other computers. If the calculation ends on the control computer, it will send the message to the other agents to stop their calculation, too. This is important for example in the case when a fixed number of generations is set up for evolutionary strategies and the computers involved have varying computing power, so that the prescribed number of generations takes different times.

Because each plugged-in computer is a separate agent, its drop-out makes no problem, even in the case when only the control computer works properly. However, any drop-out means a reduction of the calculation efficiency.

It is advisable to notice that this type of optimization can be extended also to other ones, e.g. the calculation of the maximum flow rate through the pipeline network or the search for minimum or maximum line pack in the network.

**CLIENT-SERVER APPROACH**

A computer program called **Client** (master) running on a single PC has a managing role to redistribute tasks to different programs called **Servers** (slave) that are running on the same or other computers. (See e.g. [3] chapter 1.3.)

This approach is similar to master slave model in [7]. Main roles of a client program are to start server programs, control and finish them after all computations. In the ES algorithm, the roles are to initialize population, distribute tasks to servers (to compute suitable offspring), receiving and storing returned results and use obtained results to generate new tasks for servers.

The roles of each server program are to receive, store and distribute obtained tasks to computation engines (available at particular machine) and resolve exceptions and manage reports from engines, aggregate computed results to blocks and send them back to the client. In each server there could be active several computation engines (the number corresponds to CPU cores numbers). A computation engine processes tasks that means it generates an offspring from a given parent, sets its parameters into the network, computes the simulation and returns a value of an objective function (or a penalization value).

**Advantages:** All available data is concentrated on a single computer enabling solution of a task with higher complexity of processing or evaluation. In the case of population methods, centralization allows to freely adjust the population size and process the population as a whole. Data can be easily analyzed and archived (this is especially important during development). You can maintain information about available servers, their vitality and current performance. If a server fails, it has no fatal consequences for running the calculation. Some modifications of a client part can be developed without the need to create and distribute a new installation on all computers.

**Disadvantages:** If a client fails, the calculation cannot continue. The time between the generation of task and the usage of its result is prolonged due to the procedural distance between the client (manager) and servers. The client is supposed to process and generate assignments for all computers, so it must be fast enough. This limits maximal number of servers (size of computer cluster). Fortunately, the more computationally intensive calculation on a single entry, the larger cluster size is allowed.

It was necessary to choose communication protocols, to propose how to distribute work and aggregate results and how to balance load on servers.

TCP / IP (Transport Control Protocol / Internet Protocol, see [3]) is more complicated and has some problems with Windows OS implementation. However, it guarantees higher reliability and lossless connectivity than UDP / IP (User Datagram Protocol / Internet Protocol), which does not guarantee connectivity. TCP makes it possible to connect to arbitrary computer with a specific IP (Internet Protocol) address by creating a synchronized shared archive between computers that can send essentially unlimitedly long messages (by directional TCP also the connection with remote computer is possible Fig. 4). On the other hand, UDP Broadcast is omnidirectional, has an easier implementation, but only spreads within the Broadcast domain (local part of the network up to the router see Fig. 2).

Since we have chosen a centralized approach, it is better to use the implementation-intensive directed TCP communication instead of broadcasts. In addition, this option allowed us to experiment with computers that are outside the local broadcast domain. Moreover, it allows to experiment with virtual machines at cloud service providers, e.g. Google Cloud service.
Even after selecting a communication protocol, there is still a problem of how to split the entries and send them to the server as individual parts. The distance between the manager and executor(s), the heterogeneity of the computer cluster, as well as the dependence of the complexity of the particular calculation on the task entry, cause a significantly variable delay. This, in the case of block messaging, leads to inefficient use of server computing resources. (The tasks are somewhere accumulated and elsewhere missing.) That is why we preferred the strategy of continuous flow of assignments and results. For the sake of communication efficiency, these data streams are executed asynchronously as smaller blocks. While on the client and server side, buffers are prepared in advance to receive huge amount of data abruptly. They are able to collect these exchange data blocks and send them back in smaller parts for processing, respectively, for further aggregation.

The algorithm begins with randomly generated assignments which are sent to servers for computation and the computed results are returned back to the client. From all randomly generated and successfully computed values is created the “File with Assignments”. Procedure describe in next sections is visualized in Fig. 5. In a standard iteration the “File with Assignments” is read by blocks and each block is moved to “Assign buffer”. From there, tasks (if there are some) are distributed to servers, where they are stored in “Input buffer”. Tasks located in the “Input buffer” are then locally distributed as small batches to “Computation Engines”. For each assignment at a computation engine an offspring is generated and evaluated (if it is not successful the process is repeated predefined number of times, the treatment of unsuccessful generation depends on the chosen strategy of an ES algorithm).

After successful creation and evaluation of an offspring the results are accumulated into local “Output buffer” and are sent as small blocks to the client, where these are stored in a “Result buffer”. The content of the “Result buffer” is periodically stored into the “File with results” (where only successfully generated offspring are stored). If all tasks from the “File with assignments” are assigned for computation (no more tasks remain) then all results from the “Result buffer” and the “File with results” are used to generate new assignments which are stored in the “File with Assignments”. After that the next iteration starts. Whole process is repeated until predefined number of generations is computed or until overall computation time is reached.

By default, mostly for historical reasons, computers with different architectures, CPU cores, operating systems, and various usage modes are connected to the local network. While computing, the servers are freely available for their original purpose since only their residual performance is used for the cluster aims. If you want to use their entire installed power, you need to do it when they are out of service (e.g., out of business hours). Otherwise, you can use only such part of computing resources that avoid limiting the primary user.

Computers’ load should be adequate to their current free performance. If the server is late, it is advisable to inform the client to slow down or stop sending additional entries.

It turned out that the application load control itself is very problematic in Windows OS. Any effort to change the load, based on the current use of the PC, competed with the operating system task scheduler. In certain cases, the estimate of free resources did not correspond at all to the actual amount of resources that could be used. That is why we only set a lower application priority and left the server load control on the operating system’s task scheduler.
It was only necessary to determine how many entries the client should send to each server. Our experiments were conducted in a controlled environment, so we could do stress tests using a series of simulations and compile a table of maximum number of entries that the server ideally handles over a defined time. To keep the server busy, it is reasonable to keep this maximum number of entries. The easiest way to do so is: whenever a computed entry from the server is received, a new entry is sent to it. If a server slows down, sending new entries will naturally slow down and vice versa.

**STEADY STATE INDICATION**

In transient optimization, we are looking for a transition to a determined stationary state. So there is a question of how to identify that the simulation has actually passed into this stationary state.

One possibility is to test how close a stationary state has been approached, regardless of whether it is the desired final state. In this case, it is implicitly assumed that the simulation is unsteady throughout the transition and the only stationary state to which the simulation can be approached is the final one. Such a procedure requires a stationarity test and can be based on monitoring the thermodynamic state of the gas:

- **at selected locations on the network for a preset amount of time**
  - It is suitable to monitor the pressure and the inlets to the node (or at other location on the pipe).
  - The temperature is not critical during the transition. It slows down tardily therefore it need not be controlled as much.
  - A sufficient delay between an observation start and the last control event is necessary, so that the influence of the control can be noticed at the observed places.
  - Due to observations, the simulation time should be extended. (The extension of the simulation time depends on the size of the network and the distribution of selected checkpoints).

- **along the network (all pipes) at the current time**
  - It is advisable to monitor the flow along the pipes since this is constant in a steady state.
  - Pressure and temperature are not suitable because they change along the pipe even at stationary conditions.
  - The comparison can be made just after the last intervention because the influence of the control in the nearest piping around the control element (compressor station) is very fast.
  - There is no need to extend the simulation time. It will save computational power.

- On short or blinded pipes with great discretization, the simulated flow can unrealistically oscillate and this can distort the resulting rate of stationarity. A solution is to exclude short pipes from analysis.

In agreement with this summary, we decided to measure the transition end using the steady state indicator (SSI), based on observation of the gas thermodynamic state along the network (all pipes) at the current time. The SSI was calculated from the absolute value of difference between the maximum and minimum flow on each pipe (either in the endpoints or in several measuring points along the pipeline) and the resulting value is determined as the maximum from these differences. By comparing this value with a predetermined threshold, it is determined whether the state of a simulation is close to the desired steady state or not.

$$SSI_{\text{threshold}} > \max_{n \in \text{pipes}}\left(\max_{i \in \text{points}}\left(\frac{Q_{\text{sim},i}}{Q_{\text{end},i}}\right) - \min_{i \in \text{points}}\left(\frac{Q_{\text{sim},i}}{Q_{\text{end},i}}\right)\right)$$

($Q_{\text{sim}}$ - mass flow from the simulation.) These tests are in most cases effective. But the measurement points' sparseness as well as the short time since the last intervention or the short observation period are sources of uncertainty that leads to false positivity. In addition, if the linear network is composed of short pipes, a very small flow difference can be achieved between the pipe ends, but these small differences together create a big difference between the outflow and inflow nodes of the network. As the maximum flow difference often occurs just between pipe ends, this phenomenon cannot be eliminated by increasing the number of monitored points along the pipe.

Another approach to the transition termination end detection calculates a rate of similarity between the simulation and final stationary state. In this approach, the known final steady state is directly used. Although this approach is more complicated, it increases the reliability of the test. We mainly dealt with three approaches, which measure the difference between the dynamic parameters of the simulation and the corresponding target values.

Comparing the simulated and final steady state

- **flows along the pipe**
  - When calculating the indicator value, it is not necessary to process each pipe separately, but just go through all measuring points on all pipes and determine the maximum difference of the current state from the desired one.
  - $$SSI = \max_{i \in \text{all points}}\left(\frac{Q_{\text{sim},i}}{Q_{\text{end},i}}\right)$$
  - ($Q_{\text{sim}}$ - mass flow from the simulation, $Q_{\text{end}}$ - mass flow from the final state)
  - The target flow is constant along the entire pipe, it simplifies implementation.
o A short pipe network is eliminated and increasing the number of measuring points increases the test sensitivity.

- pressures along the pipe
  o When comparing simulated and stationary state, there is no problem of pressure variation along the pipe. Calculation of SSI value is analogous to computation from flow.
  o Stationarity test defines constraint of working area for optimization. If SSI and contracted pressure is tested simultaneously, it may be advantageous use pressures along pipelines to compute SSI instead of mass flow rates. In this case, the both constraints can be implemented as penalties in the same physical units.

- accumulation of gas in pipes
  o The accumulation difference value is more stable than flow or pressure differences at selected points along the pipe.

Parameters compared between a simulation state and the final steady state can be organized as a real valued vector and a measure (a distance) can be computed by arbitrary vector norm. It is possible to use, e.g., infinite norm (maximal absolute value of difference of parameters through all coordinates), Euclidean or Manhattan distance.

To determine condition of achieving of final steady state is independent hard problem. The proposed procedures have many pros and cons. This problem requires further investigation.

**ES MODIFICATIONS**

ES algorithm benefits from variability of population. If the population in ES is very small or the ratio between parents and offspring is too low (maximal value could be 1), then the variability is insufficient and the method is not able to search parameter space enough. This problem occurs mainly in highly dimensional problems (dimension > 10). Besides, these methods were developed for unconstrained optimization thus the classical implementation has problems to effectively search the parameter space near constraints and mostly prematurely converges. Simple modification of the method when multiple attempts are used to generate an offspring slightly reduces this problem, but does not completely remove it. [4].

First few numerical experiments of transient consumption optimization show that minimum is located in a shallow region. With increasing the number of control events the shallowness emphasizes. From point of view of practice, it is not possible to choose one best solution because there is more than one control which resolves to very similar value of objective function.

All mentioned problems inspire us to modify standard ES to improve exploration and covering of shallow area around minimum. Modification also eliminates complication near constraints and provides convergence.

In the following it is assumed that parameters of an objective function are normalized to [0, 1] interval.

**Improved local exploration:** Generation of an offspring starts with the initial maximal standard deviation (std.) = 0.2 and it gets smaller if generated offspring is outside of working area. The diminishing factor is about a half. After 10 attempts it gets to one thousandth of the initial std. Therefore the maximal detail is limited, but for practical use this strategy provides sufficiently accurate results. This modification supports only the exploration properties and disregards convergence. Due to this it is not possible to use such modification alone and other mechanism which provides convergence is necessary.

**Improved elitism:** Standard evolution strategies are modified in such way that one of generated offspring is forced to have the value of an objective function the same or better than its parent. This modification support convergence and it improves optimization method in combination with “Improved local exploration” modification.

**Spatial filtration:** Standard evolution strategies are modified in such way that parents are selected not only by the value of their objective function but also by spatial context (Best individuals are chosen and all other neighbors within a certain radius (0.05) are removed. After that the next best individual of remaining population is selected and process of elimination is repeated.)

Numerical experiments presented in the chapter Results of Basic Experiment illustrate proposed modifications. “Improved elitism” and “Improved local exploration” alone do not successfully improve method, but when all modifications were utilized together, the better results were obtained. All these modifications together allow better examination of distribution of individuals, which have similar values of an objective function.
CONSTRAINTS IMPLEMENTATION

Some constraints are hard: if they are exceeded, the calculation (simulation) can no longer continue (e.g., surge crossing on a compressor) and vice versa, some constraints are soft: the calculation (simulation) can continue (e.g. the desired final state or an end pressure has been lower than contracted).

Some constraints can be tested at initialization of an objective function calculation (e.g., min., max. revs or min., max. time for control), others must be continuously checked during the simulation run (e.g., surge, choke and compressor blade temperature; max., min. pressure on pipelines or surpassing the contracted pressure at the outlet of network) and some can only be tested at the end (e.g. reaching the final steady state, keeping the average contracted pressure during the transition time).

The simulation phase is the most computationally demanding, so it is good if all possible tests are performed before the simulation and in case of a problem, it is better to re-generate or correct the problematic assignment to a task. The hard constraints during the simulation run must be strictly adhered to and therefore the constraint boundary is accessible only within the working area. This should be mainly ensured by the “Improved local exploration”. If a hard constraint is surpassed very soon after the start of simulation, the interruption of the simulation saves a lot of computational time (e.g., the computational demands of one completed simulation can be greater than 10 attempts that have been stopped below one tenth of the simulation time). Consequently, if we need to search around the hard constraint, it is better to risk a not evaluation than slow approach to the constraint from the side of the working area. Evaluation of a simulation is computationally simple, but includes verification of some constraints. Fortunately, all these constraints are soft so the result of a simulation that has reached its end can always be useful.

We have tried a number of procedures, such as modification of an objective function by adding a penalty that depends on how much the threshold has been exceeded. This procedure is not entirely desirable. Since the distortion of the objective function by a penalty can move the position of the optimal solution outside the allowed area.

It seems better to left the penalizing function separated from the objective function, and a parent who crosses the constraint will only be forced to minimize the penalty function. Thus, the procedure is analogous to that of the “Improved elitism”, except that in the work area, the individual is governed by an objective function and outside the working area by a penalty function. The procedure does not have problems resulting from the sum of objective and penalty functions. It extends the work area to a penalty area, which increases the likelihood of success in initializing individuals at the start of the optimization calculation.

Thus, the overall optimization process will consist of three parts: first, randomly generated individuals who are in the extended area (working area + penalty area), second, search for any individual in working area, third population optimization within working area.

In the case of one soft constraint, the design of the penalty function is simple. It is sufficient to return zero for an allowed value and in the case of failing constraint to return the absolute value of the difference between the value reached and the limit value (no additional scaling is required for a single constraint). For two constraints, the penalties from one and the other constraint must be combined. Ideally, one penalty can be scaled in such way that they both have the same effect on population scattering and then subsequently to choose the maximum of both as the total penalty.

BASIC EXPERIMENTS

If a simulation is very short (real computational time is under 0.5 s) latent processes (e.g., waiting for GUI common resources) could lead to ineffective utilization of workstations (servers). This can be overcome, if the number of computational engines is greater than the number of available CPU cores in a particular workstation. In case of short simulations the size of input buffer has to be minimally twice the number of computation engines.

Basic testing problem consists of a linear network with two compressor stations (CS1, CS2) each with a single compressor for simplicity and easier interpretability of achieved results. Compressors have defined range of allowed revolutions (min. and max. revs), surge and choke limits and also limit for temperature of blades. Input node, CS1, CS2 and output node (delivery point) are interconnected by pipelines 50km (31.07 miles) long and 1.2m (21.43 in) in diameter.

The goal is to find such control of compressors (revs trends for CS1 and CS2), where after a defined time (8h) the network state changes from the initial steady state with mass flow 400 kg/s (1713 MMSCFD), see Fig. 6, to the given final steady state with mass flow 300 kg/s (1285 MMSCFD), Fig. 7, with use of minimal possible amount of technical gas.

An achievement of the final steady state is tested by comparing simulated mass flow with mass flow from final steady state along all pipelines and at five points on each pipeline. It is computed maximal value of absolute values of all differences and this value must be smaller than assumed threshold (0.5 kg/s = 2.21 MMSCFD). It is also demanded that average value of the output pressure in output node is above the contract pressure (5 MPa = 725.2 psi), i.e., locally it is possible to violate this condition, but at the end of simulation overall balance must be above the contract pressure.
The initial and final states have been computed by steady state optimization using standard Evolution Strategies. It turned out that both states are very close to or even on a technological limit (in these cases, the limit for choke and temperature of blades).

Fig. 6 Initial (starting) steady state with outflow 400 kg/s (1713.5 MMSCFD) and corresponding compressor maps with depicted working points for CS1 a CS2.

Fig. 7 Final (ending) steady state with outflow 300 kg/s (1285.1 MMSCFD) and corresponding compressor maps with depicted working points for CS1 a CS2.

The control of each compressor is defined just by two events. Each event consists of the time component and the value component representing revs value (see Fig. 8). The first event has as parameters both components, the second event has as a parameter just the time. Its value component is taken from final steady state. Due to this there are 3 parameters for each compressor and transport network contains two compressors, thus there is 6 dimensional parameter space. We assume that parameters are normalized to [0, 1].

Fig. 8 Compressor control is based on events with determined time and target revs (average change of speed is 10 rpm/s).

Fig. 9 Three best controls of CS1 and CS2 for consumption optimization where all proposed ES modifications are used and jump in outflow is from 400 kg/s (1713.5 MMSCFD) to 300 kg/s (1285.1 MMSCFD).

In Fig. 9 there are presented few revs controls which provide minimal consumption in 8 hour time period. As you can see also with two control events the solution was not trivial (see Fig. 9). CS1 decrease revs even below ending revs at the 20 min (minimal control time) and after 3 hour and 30 min it achieved final revs. The CS2 changed revs to ending value event after 6 hours. It can be interpreted that the CS2 works also for CS1 and the CS1 can work in economic mode for a while what minimizes total consumption.

EXPERIMENTS WITH MODIFICATIONS OF EVOLUTION STRATEGIES

Problem is analogous to previous chapter. In this case the goal is to find such control of compressors (revs trends for CS1 and CS2), where after a defined time (10 h) the network state changes from the initial steady state with mass flow 350 kg/s
(1499 MMSCFD) to the given final steady state with mass flow 370 kg/s (1585 MMSCFD) with use of minimal gas consumption.

Four experiments were performed:

**Standard evolution strategies (blue):** 5000 offspring, 1000 parents; initial standard deviation of an individual (std.) = 0.2; 50 repeating of experiments to generate an offspring. (No other modification has been used.)

**Improved elitism alone (black):** the standard evolution strategy experiment is extended only by “Improved elitism”.

**Filtration alone (green):** the standard evolution strategies experiment is extended only by “Spatial filtration”.

**All modifications (red):** the standard evolution strategies experiment is extended by “Improved elitism”, “Spatial filtration” and “Improved local exploration”.

Comparison of sorted values of an objective function for all members in the final population which was generated by a certain modification is depicted in Fig. 10. Std. values of all members are in Fig. 11. In Figs 10 and 11, the order number of each individual is normalized to [0, 1] for easier comparison of large populations. Fig. 12 depicts the final population members, where on x-axis is distance from the selected point in the parameter space ([0, 0, …, 0]) and on y-axis there is the value of an objective function.

![Fig. 10](image1.png)

**Fig. 10** Comparison of objective function values of individuals of final populations obtained by four experiments.

In case of standard evolution strategies, whole population converged to few local points with substantially better values of an objective, thus whole population has achieved low values of an objective (blue curves in Figs. 10 and 11) The method has fast convergence, it manages to compute more than 20 generations, but most of the individuals have small std., so they have almost no changes of position and value of objective function at the end.

![Fig. 11](image2.png)

**Fig. 11** Comparison of standard deviation of individuals of final population obtained by four experiments. (y-axis does not have units because the normalized parameter space is used.)

If we use standard evolution strategies with the “Improved elitism” (black curves in Figs. 10, 11 and 12) the population does not have enough time to converge. Large amount of the final population has rather big stds. (black curve in Fig. 11) and worse objective function values (Fig. 10). This modification manages to compute less than 10 generations. This can be explained by the fact that for computation of improved individuals is used rather large amount of tries (50), but there is very little chance that with a large sigma (std.), it can be find an offspring with better objective functions value.

![Fig. 12](image3.png)

**Fig. 12** Comparison of clusters of final populations obtained for different experiments. Colors correspond to the previous Fig. 11. The red cluster corresponds to “All modifications” experiment which covers the area around minima better than other resulted clusters.

When standard evolution strategies are coupled with “Spatial filtration”, the final population has low values of std. (see green curve in Fig. 11). However the filtration increases and conserves spatial variability of population what increases a
searching ability of the method on the other side it also restricts the convergence of population (green points Fig. 12). Procedure can provide more than 20 generation per hour. In spite of this the population cannot converge and other mechanism like “Improved elitism” is required (certainly the problem with wasting of objective function evaluation need to be solved first).

The use of “Improved local exploration” alone in principle does not support convergence. Thus experiment with this modification has not been carried out. Method with all three modifications active (red points in Fig. 12) achieves the better results than standard evolution strategies. By comparing red and blue curves in Figs. 10 and 11, it can be seen that population generated by a modified method has retained variability and some individuals are better than any individuals obtained by standard evolution strategies, which converging prematurely. In case, where there is 10 tries to generate an offspring with decreasing std by failed generations, in the middle of parameter space, large std 0.2 is used and this also enable to generate an offspring near the border (so near the border there is not necessary to use so many repetitions to generate an offspring like in standard evolution strategies, this allows to used “Improved elitism” without wasting of objective function computations). This modification is therefore fast and it manages to generate more than 30 generations in 1 hour.

**RESULTS OF EXPERIMENTS ON**

**COMPLEX NETWORK**

In case of complicated (long) simulations (real computational time is over 5 s) we have observed that for PCs up-to 4 cores the optimal utilization was achieved when the number of computational engines corresponded with the number of available CPU cores, however for PCs with higher number of cores, it has been better to use one more computational engine than number of cores. Here we do not distinguish hyperthreaded and real CPU cores, e.g., Intel Core i3 computer architecture with 2 real CPU cores and with hyperthreading switched-on is treated as PC with 4 CPU cores.

Total number of parallel simultaneously running simulations in our local network was 163 from which there were 87 in a broadcast domain at Department of Applied Mathematics. For some computations we have added additional 384 simulations from Google Cloud where we have rented 12 virtual machines each with 32 CPU cores. (We have used an evaluation period with initial credit 300 USD, the most expensive item was a license for Windows machines, which was 0.54 USD/CPU/hour) Total number of CPU cores was 547. Used computers had various numbers of CPU cores and architectures and ran on different version of Windows OS (Windows 7-10 and Windows Server Data Center R2).

![Complex network diagram](image)

In Fig. 13 there are 5 trends of revolutions on compressors in CS1, CS2 with the smallest objective function value. As it can be seen although the shapes are varying the values of objective function are very close, thus with respect to model and numeric inaccuracy it is currently hard to select the single trend as the final solution (without any additional assumptions).

![Comparison of 5 best revs controls](image)

In the final steady state the outflow was 730 kg/s (3127.06 MMSCFD). In the final steady state it was 930 kg/s (3983.80
Outflow was changed in five minutes after one hour of simulation time.

In the used network we have either joined short pipelines to longer ones or we have excluded them (mostly with one side blinded). Space discretization step was prolonged to 3 km (1.86 mile). The creation of outputs (storing the time history of a simulation into database) turned out to be very time consuming thus we have turned it off. For the control of hard constraints it is necessary to have network state update at least each 60 s of simulation time. If the stopping of a simulation (in case that some constraint is violated) and/or the running of analytical procedure, which evaluates the current SSI (Steady State Indicator) value, are computational intensive it is better to run them less often. In our case if the analytical procedure was not run then 8 h simulation lasted 26 s, if it has been run each 5 min it lasted 31 s, but if it has been run each minute the total computation time has been 74 s. For our purposes we chose 5 min simulation time evaluations of the analytical procedure.

The task was to find a control, in which the network finishes near the final steady state as soon as possible. The nearness was expressed by a value of SSI defined as maximal absolute mass flow difference between simulation state and final steady state (we check the flow only in six equidistantly distributed points along each pipeline.) We have used ES with all modifications (see chapter ES modifications) to solve it.

In our experiments we have used two limits for SSI 15 kg/s (64.25 MMSCFD) and 20 kg/s (85.67 MMSCFD). The value of a goal function was computed as interpolated time of reaching limit SSI value during simulation. If the limit SSI was not reached within 8 h simulation time, the returned value of an objective function was 8 h + achieved SSI value. (The interpolation is necessary because we used discrete checking of the SSI values, each 5 min of simulation time.)

It turned out that stability of computation of SSI in complicated networks could be improved by excluding short pipelines. In Fig. 15 there is the comparison of SSI trends for all pipelines (dashed lines) and for pipelines longer than 15 km (9.32 miles) (full lines). It can be seen that the SSI computed for long pipelines is as good as SSI computed for all pipelines.

The control of each compressor station (CS) is encoded similarly as in Fig. 8 by two parameters. The only difference from basic experiments is that time of first event is fixed to 20 min. The first parameter represents the new revs of first event and the second one represents the time of second event when revs are set to final steady state revs of a CS. We have 2 CSs thus we have a 4 dimensional task. Outflow jumps to the new value after one hour of simulation time but the control can start at 20 min. that means a preventive control could be searched. Low dimensionality might suggest that it is an easy task. However, we use simulation of a complicated network with many technological constraints. If the size of the network doubles the necessary evaluation time also approximately doubles. Between the size of a network and computational complexity there is almost linear ratio. But, increased complexity of control increases dimensionality of a task. If we assume at least 10 evaluations per dimension, rough estimate of total number of necessary evaluations for 4D task in $10^4$. This estimate is in accord with our earlier findings for steady state optimization using Evolution Strategies [6], where for 4D task there were necessary more than 20 000 evaluations of a goal function. We have benchmarked computational power of our cluster (including Google Cloud machines) and for the used network and 8 h simulation we were able to obtain 31 000 evaluations within 30 minutes. For 5D task it is needed $\sim 10^5$ evaluations. Thus solving such task within 30 min it is necessary three times larger cluster ($10^5 / 3.1 \times 10^2 \sim 3.22$).

In our experiments we disregarded the constraints on contracted pressure in output nodes. There were two reasons, the first, a time transition optimization pushes the last control to the starting time of simulation thus the necessity to keep contracted pressure does not influence the quality of the found solution significantly. The second, the tested network has 6 output nodes, which complicates the constraint checking. From experiments turned out that contract pressure condition was violated over 0.3 MPa (43.51 psi) for the time shorter than 30 min of simulation time.

The limit SSI was expected to influence the state of the network near final steady state. For example 1 kg/s (4.28 MMSCFD) limit SSI was supposed to decrease transition time where this limit was achieved. In our experiments we have found out that the transition time can be reduced even if the limit SSI is set to larger values (e.g., 15 kg/s (64.25 MMSCFD), 20 kg/s (85.67 MMSCFD)). By relaxing limit SSI to larger values the workspace for an algorithm is larger thus it easier for a population to search the space. For very small limit SSIs (< 15 kg/s (64.25 MMSCFD)) the space is too restricted and population stagnates.

The results of the experiment are summarized at Figs. 15 and 16. Fig. 16 depicts controls with best transition times for limit SSI 15 kg/s (64.25 MMSCFD) and 20 kg/s (85.67 MMSCFD) obtained after 30 min. of total computation time. Both solutions have the same character, first the revs are increased for a while for to move an accumulation and then the revs are set to final steady state revs. The transition time is shifted to the start of simulation.

The best solution (dark green curve in Fig. 15) achieved SSI limit 20 kg/s after 2 h. and 5 min and the limit 15 kg/s after 2h and 15 min. This result (green curves in Figs 15 and 16) is substantially better than result obtained for trivial control of changing the revs to final values on both compressors at 20 min. after start (red curves in Figs 15 and 16).
The calculation of the purpose function itself is computationally complicated in transient optimization because of the complexity of pipeline network, the large number of technological constraints and the length of simulation that is necessary for computation. Furthermore, the complexity increases with the number of parameters encoding the searched control.

To not substantially simplify the task, it is necessary to obtain as many computational resources as possible and to parallelize the optimization calculation. Population stochastic methods allow natural parallelism to calculate the objective function. On the other hand, this degree of parallelism is also beneficial because our simulator was developed for a single core CPU and its internal parallelization would require a non-trivial effort.

Simulations can be calculated on multiple cores of one computer in parallel and simultaneously, these calculations can be performed on multiple computers. We worked first with computers in our immediate vicinity (in a broadcast domain) and later we started to use multiple computers across the building and finally we tried to add virtual instances on Google Cloud to our local computer cluster. Overall, we were able to link 547 cores (original + hyperthreading cores), allowing us to perform (at the same time) 547 simulations (each of which lasted tens of seconds).

This common computing power has allowed making calculations with a complex network with a total pipeline length of about 2290 km, taking into account the technological constraints. However, it has been shown that increasing the number of control interventions and the number of controlled or controlling compressors very quickly complicates the task to a level where even the common accumulated power is insufficient.

The solution can also be used for complex networks, but the number of optimized parameters should not exceed 10. The proposed approach allows reducing optimization tasks lasting without parallelization more than 11 days to 30 minutes. Combining the computing power of a local cluster with external resources for calculation of critical tasks, there will be still possibility to use at least the computing power of a local network (own resources) in the case of outbound internet connection damage. This increases the reliability of the solution.

During the development two different approaches have been tried: The Client-Server approach and the collaboration of parallel running agents (smaller populations) working in parallel and sharing the best results. We started with latter applying it first to steady state optimization tasks and then to transient ones. Later during numerical experiments with transient optimization it turned out that if suitable parameters for each of the approaches are set, the results are comparable. For example, the case of isolated 25 populations sharing the best results (individuals) each population consisting of 120...
parents and 200 offspring provides similar results as the use of a single large population with 1 000 parents and 5 000 offspring.

REFERENCES

https://doi.org/10.1023/A:1015059928466

https://www.onepetro.org/conference-paper/PSIG-1011


http://dx.doi.org/10.1155/2010/185063

https://www.onepetro.org/conference-paper/PSIG-1810

https://doi.org/10.1515/tmmp-2016-0024

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1. 18.924

https://www.onepetro.org/conference-paper/PSIG-1319

https://www.onepetro.org/conference-paper/PSIG-1806

ACKNOWLEDGEMENT

This research has been supported by the Grant agency VEGA, project 2/0153/16. We also acknowledge the use of Google Cloud when performing experiments on a complex network.

AUTHOR BIOGRAPHY

Igor Mračka is a postdoctoral research fellow at the Mathematical Institute of the Slovak Academy of Sciences, Bratislava, Slovakia. He has attained his PhD degree in applied mathematics from the Comenius University, Bratislava, Slovakia. Currently, his research focuses on gas transport simulations, leak detections and locations, signal processing and neural networks.

Tibor Žáčik is a senior research fellow at the Mathematical Institute of the Slovak Academy of Sciences, Bratislava, Slovakia; there he has attained his PhD postgraduate academic degree in global analysis. His main subjects of interest in the field of gas transport are gas simulations, leak detections and locations, and transport optimization problems. He has more than 15 years of experience in this field.

Marek Hyčko is a research fellow at the Mathematical Institute of the Slovak Academy of Sciences, Bratislava, Slovakia. He achieved PhD. degree at the Mathematical institute in probability and mathematical statistics. He has many-years experience with statistical analysis and data processing.

Rudolf Hajossy is a senior research fellow at the Mathematical Institute of the Slovak Academy of Sciences, Bratislava, Slovakia. He has 50 years of experience in physics—its education and its application in electrical engineering and power industry, measuring technology, medicine, biomechanics, meteorology, safety and security, modeling and optimization of gas transport. He has a PhD degree in experimental physics from the Comenius University, Bratislava, Slovakia.

Peter Somora has 13 years of experience in the gas transport optimization and leak detection. He has attained his PhD degree in mathematical analysis and differential equations, acquired at the Mathematical Institute of the Slovak Academy of Sciences. He is interested in gas simulations, GPGPU computations, leak detections and locations, compressor modeling, etc. He is a member of the Slovak Gas and Oil Association.