ELSEVIER

International Conference on Computational Science, ICCS 2010

Integer simulation based optimization by local search

Jaroslav Sklenar[a]*, Pavel Popela[b]

[a]*University of Malta, Msida, Malta*
[b]*Brno University of Technology, Brno, Czech Republic*

Abstract

Simulation-based optimization combines simulation experiments used to evaluate the objective and/or constraint functions with an optimization algorithm. Compared with classical optimization, simulation based optimization brings its specific problems and restrictions. These are discussed in the paper. Evaluation of the objective function is based on time consuming, typically repeated simulation experiments. So we believe that the main objective in selecting the optimization algorithm is minimization of the number of objective function evaluations. In this paper we concentrate on integer optimization that is typical in simulation context. Local search algorithms that try to minimize the number of objective function evaluations are described. Examples with both analytical and simulation-based objective functions are used to demonstrate the performance of the algorithms.

Keywords: simulation, integer optimization, local search

1. Introduction

Any deterministic single objective optimization problem can be expressed as $\min\{f(\mathbf{x}) \mid \mathbf{x} \in S \subseteq \mathbb{R}^n\}$ where f is the scalar multivariate objective function, n is the problem dimension and S is the set of feasible solutions. We note that maximization of f is minimization of $-f$. The set S is of course problem dependent, but often (not always) it can be expressed in terms of vector multivariate functions $\mathbf{g}$ and $\mathbf{h}$ representing the inequality and the equality constraints: $S = \{\mathbf{x} \in X \subseteq \mathbb{R}^n \mid \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \mathbf{h}(\mathbf{x}) = \mathbf{0}\}$ where the set X further restricts the acceptable values, for example it can be a set of integer vectors in case of integer problems ($X \subseteq \mathbb{Z}^n$) or a combination of continuous and integer variables for mixed problems ($X \subseteq \mathbb{Z}^m \times \mathbb{R}^{n-m}$). There are very many optimization algorithms that have been developed to solve the above problem. For their overview and the background theory see for example the book [1] to mention one out of many. A common assumption of all optimization algorithms is that for a given solution vector $\mathbf{x}$ the functions f, $\mathbf{g}$ and $\mathbf{h}$ can be evaluated. This is true for all textbook examples, but unfortunately in reality the situation is different. For many especially stochastic systems we know that certain functions exist, but no algorithms are available for evaluation of these functions. Then the only option is approximation and/or simulation. Here we

concentrate on simulation-based optimization that compared with classical optimization theory brings its specific problems and restrictions. Function evaluation by simulation is in fact sampling where the population is made of all possible realizations of some function $F(\mathbf{x}, \xi)$ where $\mathbf{x}$ is the solution being tested and the random vector ξ represents the random numbers generated during a particular simulation experiment. Note that the vectors ξ used in the experiments don't necessarily have the same length. We design the simulation experiment in such a way that the population mean $\mu = \mathbb{E}[F(\mathbf{x}, \xi)]$ is close to the function $f(\mathbf{x})$ being evaluated. Similarly for the components $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$ of the vector constraints functions. By repeated simulation with changing seeds of random generators we obtain a certain number r of samples and we compute the sample mean that we use instead of the exact unknown function value:

$$f(\mathbf{x}) \simeq \overline{X} = \frac{1}{r} \sum_{i=1}^{r} F(\mathbf{x}, \xi^i) \tag{1}$$

It is well known that the expectation of the sample mean $\mathbb{E}[\overline{X}]$ and the population mean are equal, so by increasing the number of experiments the sample mean $\overline{X}$ computed by the above formula can approximate the actual population mean arbitrarily close. The population variance $\sigma^2 = \mathbb{V}\mathrm{ar}[F(\mathbf{x}, \xi)]$ can be decreased by increasing the duration of the simulation experiments, the sample variance $\mathbb{V}\mathrm{ar}[\overline{X}] = \sigma^2 / r$ can be decreased by increasing the number of experiments or by applying any other variance reduction technique. These problems represent standard simulation topics dealt with in most simulation textbooks. See for example the book [2] to mention one out of many. So for the purpose of this paper we assume that the objective function $f(\mathbf{x})$ and the component functions $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$ can be evaluated by repeated simulation experiments with required precision.

Simulation based optimization was studied by many authors. Also various approaches have been taken. Most methods assume continuous variables and try to apply algorithms of classical optimization theory. For example great effort has been invested to find the approximate values of gradients by simulation. Methods are based on a common idea: evaluate the function at perturbed points and use the differences to compute the gradients approximately. For more see for example [3]. These methods suffer from the fact that close to the minimum the norm of the gradient can be very small, much smaller than the simulation error. Thus the results may be completely misleading, for example we may obtain wrong minimization directions. That's why robust algorithms based on the function values only are probably more promising than methods based on approximated derivatives. For a more detailed discussion about simulation-based optimization see the book [4] and papers [5], [6], [7] and [8] among others. Next we discuss the problem of feasibility in simulation-based optimization context.

2. Feasibility in simulation context

Because of obvious reasons discussed above simulation does not provide exact results. We can increase the number and duration of simulation experiments and apply variance reduction techniques, but this can only decrease the error, not eliminate it. This simple fact makes the use of equality constraints such that the functions $h_j(\mathbf{x})$ are evaluated by simulation very problematic. The same applies to tight inequality constraints. The result is that we have to accept a certain tolerance on equality; otherwise the solution vectors would all be infeasible with respect to equality constraints and possibly infeasible with respect to tight inequality constraints. This applies generally even to constraints whose functions can be evaluated without simulation. The difference is the value of the tolerance that in case of known functions is under user's control as one of general tolerance related optimization parameters of the tool used (see for example the Optimization toolbox of Matlab).

Another problem related to feasibility is the fact that infeasibility may not be revealed before starting the simulation. So it may happen that we simulate for example unstable queuing systems where crash sooner or later inevitably happens. Thus it is very important to try to check feasibility before actually starting the simulation. Of course it may be impossible. Then the simulation model should contain tests to detect infeasibility. For example we can define an upper bound on some effectiveness parameter like queue length or total number of customers in the system and stop the simulation if it is reached. A general guide hardly exists; we have to make use of any knowledge available about the system simulated.

So far our assumption was that feasibility is tested by the values of the functions g_i and h_j. Alternatively, depending on the optimization algorithm used, it is possible that the simulation experiment reports infeasibility by a very large objective function value. Programming languages mostly allow working with "infinite" values (*maxreal* in SIMULA, POSITIVE_INFINITY in Java). We then solve an unconstrained problem for which only evaluation of

the objective function *f* is needed, provided the optimization algorithm copes with possibly infinite objective value. Local search algorithms discussed next have this capability.

3. Integer optimization

From now onwards we concentrate on integer optimization problems because many practical decision making situations can be formalized as integer programming problems, for example selection among a certain number of alternatives and many configuration selection problems for which integer decision variables are typical. Solving (mixed) integer optimization problems is generally much more difficult than solving problems in continuous variables. Many algorithms have been developed; see for example the book [9]. Most integer programming algorithms are based on solving a problem relaxation with dropped integrality constraints and a procedure eliminating non-integer solutions until an integer optimum is reached. These algorithms are mostly useless in simulation-based optimization context because the relaxation either does not exist or it cannot be evaluated. This reduces the choice of algorithms drastically. Various deterministic and heuristic algorithms have been described. For example the paper [10] describes a method based on repeated solution of a linear approximated program whose parameters are found by simulation. Often various local search methods are used. It has been reported that in spite of the primitive basic idea of local search methods, they have found many successful application areas. The methods are generally very robust because they are based on function evaluation only. There are no assumptions about the functions involved that is very important in simulation-based optimization where we know practically nothing about the function except the fact that it exists. They also cope with infeasibility reported by a large "infinite" objective value together with standard feasibility check before starting simulation. The disadvantage of local search methods is the possibility that very many function evaluations might be needed. We know that in simulation-based optimization the function evaluation is very costly because each evaluation represents a number of possibly long simulation experiments. The purpose of this paper is description of local search algorithms that try to minimize the number of evaluations while keeping all important advantages of local search.

Here we limit ourselves to exhaustive local search called "hill climbing" for maximization ("valley descending" for minimization). The basic idea is well known. We start at a given initial feasible solution and we keep moving to better points in a neighborhood until no better point exists. Then we have reached a local optimum. To implement this trivial idea, we have to make several decisions that affect very much the performance of the method. The choice is not simplified by the fact that a generally best local search method certainly does not exist. We can only compare the variants of the method with respect to particular classes of integer problems. Next we discuss some common points assuming integer problems without continuous variables. Without loss of generality we consider minimization problems.

3.1. Selecting the neighborhood

Neighborhood $\mathbb{N}(\mathbf{x})$ of a feasible point $\mathbf{x} \in S \subseteq \mathbb{Z}^n$ is a mapping $\mathbb{N} : S \to 2^S$ such that the points in $\mathbb{N}(\mathbf{x})$ are in a certain way "close" to $\mathbf{x}$. The interpretation of "being close" depends on the set S and the algorithm used. For example when solving the Travelling Salesman problem, the feasible set is made of all vectors that represent cycles and neighborhood of a cycle is made of cycles obtained from $\mathbf{x}$ by some rules – for example by replacing edges (*a,b*) and (*c,d*) by edges (*a,d*) and (*c,b*). Next we assume that neighborhood is based on geometrical distance in $\mathbb{Z}^n$. We define two discrete neighborhoods as follows:

$$\mathbb{N}_c(\overline{\mathbf{x}}) = \{\mathbf{x} \mid \mathbf{x} \in S, \sum_{i=1}^{n} |\overline{x}_i - x_i| \le 1\} \quad , \quad \mathbb{N}_s(\overline{\mathbf{x}}) = \{\mathbf{x} \mid \mathbf{x} \in S, \max_i |\overline{x}_i - x_i| \le 1\} \qquad (2)$$

We shall call the neighborhoods cross and star neighborhood respectively due to their graphical shape in $\mathbb{Z}^2$ - see Fig 1. We see that in the cross neighborhood a single component can change by ±1, so it is made of $2n$ points. In the star neighborhood all components can change by ±1 in any combination, so it is made of 3^n-1 points. For bigger n the difference of the numbers of points in the two neighborhoods is significant. For $n = 10$ we have 20 points in the cross and 59048 points in the star neighborhood respectively. This might suggest that considering only cross neighborhood may speed up the local search process. Unfortunately, by using a cross neighborhood only we can miss a local optimum even for unconstrained problems. We see this situation in Fig 1 (created by Matlab) that depicts the contours of the function $f(x,y) = (x - 4.5)^2 + (y - 4.5)^2 + 1.6xy$. The point (4, 1) doesn't have a better cross neighbor, but when using the star neighborhood a better point (3, 2) exists.
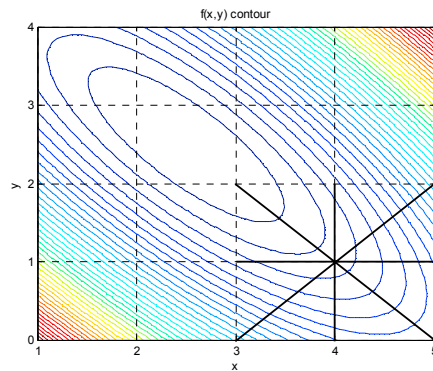
Fig 1: Discrete neighborhood of the point (4,1)

As the star neighborhood cannot be ignored it seems that there are three options:

- Use star neighborhood only.
- Start with cross neighborhood, after reaching a cross local minimum change to star neighborhood and continue until a star local minimum is reached.
- Start with cross neighborhood, after reaching a cross local minimum change to star neighborhood to find a better point. Then continue by using cross neighborhood, etc. until there is no possible improvement.

After some experimentation we believe that the second option is probably the best one in most cases. The idea is to apply the initial fast move close to a local minimum by using the cross neighborhood. Then we continue with the star neighborhood to avoid a situation depicted in Fig 1. Especially for constrained problems the cross local minimum can often be improved by additional star steps. The only problem is that the cross minimum can be far away from the true star one.

3.2. Tuning the algorithm

To implement the algorithm we have to take several decisions. First, in which order to test the neighbors of the current solution. Assuming that no further information is given, we start by cross tests at the first coordinate. This is another argument to start with the cross neighborhood because very many evaluations might be wasted until the first star improvement is found. Another decision has to be made after an improvement is found. We can either move to the first better neighbor or continue to test all neighbors and move to the best one. Testing all star neighbors would result in excessive number of evaluations especially for higher dimensions due to exponential increase of the number of neighbors with n. On the other hand testing all cross neighbors may result in better cross optimum closer to the star one, so there may be a smaller total number of evaluations because the number of cross neighbors grows only linearly with n.

Next decision is how to continue after moving to a better point. Intuitively it seems that keeping the direction of the last move might be a good choice. By direction we mean the vector made of increments. So after a move from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$, the direction is given by $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}$. We note that the components of $\mathbf{d}$ can be -1, 0, or +1. Experimentation proved that this assumption is true for unconstrained problems. With constraints it is justified when passing through the "interior" of the feasible region. After reaching the boundary of the feasible region we cannot keep the same direction. What to do in this case affects very much the total number of evaluations because very many star neighbors might be tested until an improvement is found. In other words we are trying to optimize movement along some generally unspecified nonlinear boundary. Close to corners where two or more active constraints meet we can hardly assume anything. To optimize movement along a single active boundary we follow the last direction and if this is not successful, for example due to infeasibility, we first test the so-called *close directions*. We define the set of close directions $\mathbb{C}(\mathbf{d})$ of a direction $\mathbf{d}$ as follows:

$$\mathbb{C}(\overline{\mathbf{d}}) = \{\mathbf{d} \mid \mathbf{d} \in \{-1, 0, 1\}^n, \sum_{i=1}^{n} \left| \overline{d}_i - d_i \right| \le 1\} \tag{3}$$

We note that only a single component can change by ±1 provided all are kept in {-1,0,1}. So assuming $n = 3$, close directions of (1,0,1) are (0,0,1), (1,1,1), (1,-1,1) and (1,0,0). As all directions are located on the hypercube of size 2 with centre in the origin, close directions are those whose geometrical distance from the current direction is 1. After unsuccessful tests of all close directions, we have to evaluate the remaining ones. Here again we move gradually more and more away from the last successful direction. This is implemented by converting a direction considered as a number in base 3 with shifted digits into decimal and moving up and down from this number with possible wrap around operations to keep the range $[0, 3^n-1]$. This continues until a better solution or a star local optimum is found. So for example the direction (1,0,1) is considered as the number $212_3 = 23_{10}$. Then we test the directions that correspond to decimals 22,24,21,25, etc.

The main objective is minimization of the number of function evaluations. That's why we have to make sure that each integer solution is evaluated mostly once. When using star neighborhood, there are very many common points in the neighborhoods of two adjacent points. That's why we keep a list of solutions that have already been evaluated to avoid any duplication. Searching the list in internal memory is very fast and compared with simulation evaluation the time lost is negligible. Considering the above discussion we shall test the following versions of the local search algorithm. The first star move is always chosen in the middle of the range of directions.

LS_1: Star neighborhood only without close directions.
LS_2: Star neighborhood only with close directions.
LS_3: First cross by moving to first better, then star neighborhood without close directions.
LS_4: First cross by moving to first better, then star neighborhood with close directions.
LS_5: First cross by testing all neighbors, then star neighborhood without close directions.
LS_6: First cross by testing all neighbors, then star neighborhood with close directions.

4. Examples

For testing the above versions of local search, five optimization problems have been created, three analytical and two simulation based. Problem P_1:

$$\min\{f_1(\mathbf{x}) = ((x_1 - 10)^2 + (x_2 - 20)^2 + (x_3 - 30)^2)^{1.5} - ((x_4 - 40)^2 + (x_5 - 50)^2 + (x_6 - 60)^2)^{0.5} \mid \mathbf{1}^T\mathbf{x} \le 180, \mathbf{x} \ge \mathbf{0} \}$$

Problem P_2:

$$\min\{f_2(\mathbf{x}) = (x_1 - 10)^2 + (x_2 - 20)^3 + (x_3 - 30)^4 + (x_4 - 40)^2 + (x_5 - 50)^3 + (x_6 - 60)^2 + (x_7 - 70)^4 \mid \mathbf{1}^T\mathbf{x} \le 250, \mathbf{x} \ge \mathbf{0} \}$$

Problem P_3 is similar to P_2, the only difference is the sum of components $\mathbf{1}^T\mathbf{x} \le 150$ to get a constrained minimum.

For simulation we consider a hypothetical queuing network made of 4 stations shown in Fig 2. Arrival to the network is Poisson with the rate $\lambda = 1$ customer per minute. The stations are multi-channels with mean service times 3, 7, 5 and 6 minutes respectively. The actual distributions are triangular with minimum value mean - 50% of the mean, most likely value equal to the mean and maximum value mean + 50% of the mean. After service in station 1, 60% of customers proceed to station 2, 40% of customers proceed to station 3. After service in station 2, 40% of customers proceed to station 3, 30% proceed to station 4, the others leave. After service in station 3, 70% of customers proceed to station 2, 15% proceed to station 4, the others leave. Stations 2 and 3 are visited mostly once, so customers who were served by stations 1, 2 and 3 or 1, 3 and 2 all either leave the network or proceed to station 4, both with the same probability. We have 22 personnel (service channels) available that should be allocated in such a way that would minimize the total average number of waiting customers.
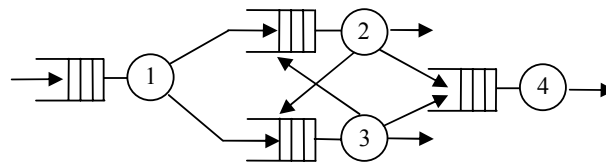

Fig 2: Example queuing network

So with all the parameters mentioned above fixed we solve the following constrained problem in four integer variables. Problem S_2:

$$\min\{L_Q(c_1, c_2, c_3, c_4) \mid c_1 + c_2 + c_3 + c_4 \leq 22, c_i \geq 1, i = 1\ldots4\}$$

where c_i is the number of channels in station i and L_Q is the total expected number of waiting customers. Due to the general distribution of the service times and non-Markovian behavior of customers, the function L_Q can only be evaluated by simulation. A simplified version without station 4 and 20 personnel was also simulated. Problem S_1:

$$\min\{L_Q(c_1, c_2, c_3) \mid c_1 + c_2 + c_3 \leq 20, c_i \geq 1, i = 1\ldots3\}$$

Another feasibility requirement not included in the above models is stability of all stations $\lambda_{eff}/c\mu < 1$ where λ_{eff} is the total effective arrival rate to the station. To compute λ_{eff} by traffic equations for our non-Markovian behavior we would need classes of customers, so we test stability by simulation.

The simulation models were written in SIMULA by using the tool QUESIM – see [13]. QUESIM supports user-friendly creation of simulation models of queuing systems. For the purpose of this paper a detailed description of the model is not relevant. Any discrete simulation tool would do provided the model can be called from the local search routine as a function that is given an integer array argument (tested solution **c**) and that returns $f(\mathbf{c})$ as the real result or "infinite" value in case of infeasibility detected during simulation. Another Boolean function should be supplied to test feasibility of **c** before starting simulation. Local search algorithms are implemented as real valued functions with 4 arguments. A typical call is: *result := localsearch3(myobjective,feasible,x0,debug,log);* where the real function *myobjective* evaluates the objective value, Boolean function *feasible* tests the feasibility of the solution before calling *myobjective*, *x0* is the starting point, Boolean *debug* turns on the debugging mode in which the function stops at each solution and Boolean *log* turns on creation of the log file with all visited solutions. Note that the local search is independent of the type of the objective function evaluation in *myobjective*. It can just return the value of an expression in case of analytical models or it can perform repeated simulation runs in case of models S_1 and S_2. Feasibility of models S_1 and S_2 is tested by lengths of queues. If any queue reaches the length 500, the system is considered unstable. Simulation is then terminated; infeasibility is reported by a large value (*maxreal* in SIMULA) of the objective function..

5. Results

The results of solving the problems P_1, P_2 and P_3 are shown in Table 1, Table 2 and Table 3 respectively. In all three cases the search started with $\mathbf{x}_0$ made of all ones. The obvious global optimum found for P_1 is $\mathbf{x}_{opt1} = (10\ 20\ 30\ 0\ 0\ 0)^T$, $f_1(\mathbf{x}_{opt1}) \cong -87.75$. The obvious global optimum found for P_2 is $\mathbf{x}_{opt2} = (10\ 0\ 30\ 40\ 0\ 60\ 70)^T$, $f_2(\mathbf{x}_{opt2}) = -33000$. The certainly not obvious optimum found for P_3 is $\mathbf{x}_{opt3} = (0\ 0\ 28\ 17\ 0\ 37\ 68)^T$, $f_3(\mathbf{x}_{opt3}) = -131810$. As this optimum was reached from various starting points we believe that it is global, though it may not be unique. We see that optima for problems P_2 and P_3 are not constrained by the inequality, optimum for the problem P_3 is. In all cases all versions of local search found the same optima.

Table 1. Problem P_1 results

Local search version	LS_1	LS_2	LS_3	LS_4	LS_5	LS_6
Total number of evaluations	943	614	280	280	786	786
Cross tests	0	0	73	73	616	616
Cross improvements	0	0	60	60	128	128
Star tests	942	613	206	206	169	169
Star improvements	152	91	0	0	0	0

Table 2. Problem P_2 results

Local search version	LS_1	LS_2	LS_3	LS_4	LS_5	LS_6
Total number of evaluations	4588	1870	1183	1183	3226	3226
Cross tests	0	0	225	225	2302	2302
Cross improvements	0	0	207	207	636	636
Star tests	4587	1869	957	957	923	923
Star improvements	401	329	0	0	0	0

Table 3. Problem P_3 results

Local search version	LS_1	LS_2	LS_3	LS_4	LS_5	LS_6
Total number of evaluations	1123	958	907	912	2067	1895
Cross tests	0	0	163	163	1695	1695
Cross improvements	0	0	147	147	515	515
Star tests	1122	957	743	748	371	199
Star improvements	317	363	82	85	2	2

We see very clearly the advantage of starting the search by using cross neighborhood in versions LS_2 and LS_3. Though starting directly with star neighborhood in LS_1 and LS_2 is not good, by comparing these two results we see clearly the improvement from using the close star directions. The difference in case of problem P_2 4588 improved to 1870 is impressive. The improvement brought by close directions is also remarkable when comparing LS_5 with LS_6 in Table 3: 371 star tests decreased to 199. On the other hand for LS_3 compared with LS_4 in Table 3 close directions in fact increased slightly the number of evaluations from 743 to 748. In all cases it was better to move to first better cross improvement and not to search for the best one ($LS_{2,3}$ compared with $LS_{5,6}$). For unconstrained problems P_1 and P_2 the cross tests found the optimum, there was no improvement by additional star tests. Constrained problem P_3 shows clearly that star tests are necessary to reach the optimum.

The results of solving the simulation based problems S_1 and S_2 are shown in Table 4 and Table 5 respectively.

Table 4. Problem S_1 results

Local search version	LS_1	LS_2	LS_3	LS_4	LS_5	LS_6
Total number of evaluations	23	23	22	22	25	25
Cross tests	0	0	8	8	17	17
Cross improvements	0	0	3	3	6	6
Star tests	22	22	13	13	7	7
Star improvements	5	4	0	0	0	0

Table 5. Problem S_2 results

Local search version	LS_1	LS_2	LS_3	LS_4	LS_5	LS_6
Total number of evaluations	82	87	89	89	63	68
Cross tests	0	0	4	4	11	11
Cross improvements	0	0	1	1	2	2
Star tests	81	86	84	84	51	56
Star improvements	8	7	3	3	3	3

For problem S_1 simulation started from (5 7 5). All versions even from various starting points found the optimum at $\mathbf{c}_{opt}$ = (5 9 6), obviously constrained, with $L_Q(\mathbf{c}_{opt}) \cong 0.52$. All runs started with randomly generated seeds. Though we know practically nothing about the function L_Q, it is very likely that this is the global optimum. There is still an improvement from starting by cross neighborhood, but as we cannot start far from the optimum due to feasibility requirements, there is no initial fast move through cross neighbors as it was with analytical functions. Using close directions made no effect and it was better to take the first cross improvement instead of testing all cross neighbors.

For problem S_2 simulation started from (4 7 4 6). All versions even from various starting points found the optimum at $\mathbf{c}_{opt}$ = (4 8 5 5), obviously constrained, with $L_Q(\mathbf{c}_{opt}) \cong 1.70$. All runs started by randomly generated seeds. The optimum is probably the global one though again we know nothing about the function L_Q. This time there is no improvement from starting by cross neighborhood. The reason is that again we have started close to the optimum due to feasibility requirements. This time using close directions made in fact a negative effect in two cases by comparing figures for LS_1 with LS_2 and LS_5 with LS_6 and once there was no effect (LS_3 and LS_4). This time the best results were obtained by testing all cross neighbors in the initial stage. Unlike for the problem S_1, all versions needed star neighbors to reach the optimum.

All simulation evaluations for both problems were made by 10 repeated runs, 60000 minutes each that means about 60000 arrivals per run. The time needed to find the optimum for problem S_2 with algorithm LS_6 was about 9 minutes (Intel Core2 T5600 1.83GHz, 1GB RAM).

6. Conclusion

Various versions of local search algorithm for optimization of integer problems were tested. Due to the simulation based optimization context the main goal was minimization of the number of objective function evaluations. A new concept of close directions has been introduced. It improved the performance considerably for two analytical functions, but in case of objective functions evaluated by simulation the situation is more complicated. There are factors that affect the number of evaluations like for example the starting point, the first selected move, etc. We cannot expect that a single version of local search performing best in all cases can be found. So the way forward might be a collection of algorithms with different features. More experimentation still has to be done. For a new problem with little or no information about the functions involved, starting with the version "first cross neighborhood by moving to first better, then star neighborhood with close directions" might be a good choice.

References

1. M.S. Bazaraa, H.D. Sherali and C.M. Shetty, Nonlinnear Programming - Theory and Applications, Wiley, 1993.
2. P. Bratley, B.L. Fox and L.E. Schrage, A Guide to Simulation – 2nd Edition, Springer, 1987.
3. J.C. Spall, An Overview of the Simultaneous Perturbation Method for Efficient Optimization. In John Hopkins APL Technical Digest, 1998, Vol 19, No 4, p. 482-492.
4. A. Gosavi, SIMULATION–BASED OPTIMIZATION Parametric Optimization Techniques and Reinforcement Learning, Kluwer Academic Publishers, 2003.
5. S. Andradottir, A Review of Simulation Optimization Techniques. In Proceedings of the 1998 Winter Simulation Conference, p. 151-8.
6. F. Azadivar, Simulation Optimization Methodologies. In Proceedings of the 1999 Winter Simulation Conference, p. 93-100.
7. Y. Carson and M. Anu, Simulation Optimization: Methods and Applications. In Proceedings of the 1997 Winter Simulation Conference, p. 118-126.
8. M.C. Fu, Simulation Optimization: Methods and Applications. In Proceedings of the 2001 Winter Simulation Conference, p. 53-61.
9. G.L Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, Wiley, 1999.
10. S.J. Abspoel, L.F.P. Etman, J. Vervoort and J.E. Rooda, Simulation Optimization of Stochastic Systems with Integer Variables by Sequential Linearization. In Proceedings of the 2000 Winter Simulation Conference, p. 715-723.
11. C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization. Algorithms and Complexity. Dover Publications, 1998.
12. L.A. Wolsey, Integer Programming. Wiley, 1998.
13. J. Sklenar, Simulation of Queuing Systems in QUESIM. In Proceedings of the 2005 European Simulation and Modelling Conference ESM2005, p. 35-7.