Noname manuscript No.
(will be inserted by the editor)

Exact and Heuristic Approaches for Directional Sensor Control

Hans D. Mittelmann · Domenico Salvagnin

the date of receipt and acceptance should be inserted later

Abstract Directional sensors are gaining importance due to applications, including surveillance, detection, and tracking. Such sensors have a limited field-of-view and a discrete set of directions they can be pointed to. The Directional Sensor Control problem (DSCP) consists in assigning a direction of view to each sensor. The location of the targets is known with uncertainty given by a joint a-priori Gaussian distribution, while sensor locations are known exactly. In this paper we study exact and heuristic approaches for the DSCP with the goal of maximizing information gain on the location of a given set of immobile target objects. In particular, we propose an exact mixed integer convex programming (MICP) formulation to be solved by a black-box MICP solver and several meta-heuristic approaches based on local search. A computational evaluation shows the very good performance of both methods.

Keywords mixed integer convex programming, metaheuristics, directional sensors, Benders decomposition

1 Introduction

Directional sensors are a class of sensors that have limited field-of-view (FOV), like surveillance cameras, infrared sensors, and ultrasound sensors. Directional sensors are becoming increasingly popular due to a wide range of applications, such as surveillance, detection, and tracking. Directional sensor control has been studied before in various contexts, and has mainly focused on coverage issues: see, e.g., [1,4,5] and the references therein. Here, we study the problem

H. D. Mittelmann
School of Mathematical and Statistical Sciences, Arizona State University, Tempe, USA
E-mail: mittelmann@asu.edu

D. Salvagnin
Department of Information Engineering, University of Padova, Padova, Italy
E-mail: salvagni@dei.unipd.it

of controlling multiple 2D directional sensors for maximizing information gain corresponding to multiple targets, as introduced in [10]. The problem can be described as follows.

We are given a list of n target on a 2D plane. The location χ_j of each target j is not known with precision, but is described instead by an a-priori distribution $\mathcal{N}(a_j, A_j)$. We are also given a list of m sensors. Each sensor i has known location s_i and can be pointed into one of K possible directions. The problem consists in choosing a direction for each sensor in order to maximize the expected information gain. The choice of the expectation of the information gain is motivated by the following:

– We want our choice to be *optimal* not for a single possible scenario, i.e., for a single assignment of each target to a location according to its distribution, but on average for all possible scenarios (hence the expectation), because we have to decide the direction of the sensors once and for all.
– The measurements obtained by the sensors are affected by errors. In other words, covering a given target with a single sensor is not enough to get perfect information on the location of the target and so there is an incentive on covering the same target with multiple sensors (the more noisy the sensors, the more are needed to get the same information gain for a given target). As such, for each scenario, we compute a posterior distribution of each target and use the information gain as a quality measure.

Note that if there were only one scenario and no measuring error, we would just direct sensors in order to cover as many targets as possible (basically a set covering problem). If we had more scenarios (according to a given prior distribution) but still perfect measures, then we would look for the best coverage on *average* (still similar to a set covering problem). Since we have many scenarios *and* measurement errors, then we use the expected information gain as objective.

Let's consider the computation of the information gain in more detail. If a target j is within the field of view of sensor i, when sensor i is pointed in direction u_i, we get the measure z_{ij}:

$$z_{ij} = H\chi_j + \eta_{ij}$$

where H is the observation model and η_{ij} is the measurement noise, assumed to be normally distributed according to the distribution $\mathcal{N}(0, R(s_i, u_i, \chi_j))$ (R is the measurement error covariance matrix). No measurement is obtained if the target is not within the FOV of the sensor. For a given scenario, the measurements from all sensors are fused in order to obtain a global estimate for each target, as a posterior distribution. Note that the posterior distribution is not Gaussian in general, and computing it exactly is not tractable. For this reason, it is approximated as Gaussian distribution $\mathcal{N}(y_j, P_j)$, where the parameters y_j and P_j are computed as:

$$P_j = \left(A_j^{-1} + \sum_i H^T (R(s_i, u_i, a_j))^{-1} H \right)^{-1}$$

$$y_j = P_j \left(A_j a_j + \sum_i H^T (R(s_i, u_i, a_j))^{-1} z_{ij} \right)$$

where the summations are done only over the sensors that generated a measurement for target j.

Given a control vector $u = (u_1, \ldots, u_m)$, the corresponding objective (based on the information gain) is then:

$$E \left[\sum_{j=1}^{n} -\log \left(\frac{\det(P_j(u))}{\det(A_j)} \right) \right]$$

For practical purposes, the expectation above can be approximated by Monte Carlo methods. More precisely, we generate several samples from the joint prior distribution of the target state, and we compute the average (over the samples) objective value for a given control action.

The outline of the paper is as follows: Section 2 describes several metaheuristic approaches to the problem, while Section 3 presents two exact methods based on a mixed integer convex formulation. Section 4 reports the computational experiments. Finally, conclusions and future research directions are drawn in Section 5.

2 Heuristic Methods

Two simple greedy approaches for the problem have been proposed in [10]. While the computational results therein show that those greedy methods do not find the optimal solution in general, the solution quality is often good and can be further improved by adding a rollout procedure [2]. However, implementing the rollout procedure is not straightforward, and the results are hard to judge performance-wise, being implemented in MATLAB. It turns out that with rollout computing times are comparable to those of our slowest exact method, see Table 3, and that without any guarantee of optimality.

For these reasons, we opted for a different approach to heuristic solutions, namely general purpose local search metaheuristics. In particular, we implemented a random restart local search algorithm (RLS) and an iterated local search algorithm (ILS) [9].

Both algorithms are built upon a standard local search (LS) algorithm, based on a natural neighborhood. Given a solution u, encoded as a vector in $\mathbb{Z}^m$ and whose components are in the range $\{0, \ldots, K-1\}$, the neighborhood $\mathcal{N}(u)$ is defined as all solutions that can be obtained by changing the direction of only one sensor: in other words, $u' \in \mathcal{N}(u)$ if and only if it differs in at most one component w.r.t. u. Clearly, the neighborhood has polynomial size, containing exactly mK solutions for each center u.

The first metaheuristic that we tried is random restart local search (RLS). The idea behind the algorithm is very simple: at each iteration the local search procedure is called from a different random initial solution, and the process is iterated until some termination criterion is met. Despite its simplicity, RLS is already a definite improvement over a pure local search approach, and it is also trivially parallelizable, an added bonus given today's computing architectures. On the other hand, it is well known that such a simple strategy is not competitive with other (more sophisticated) metaheuristics as the search space grows [9].

The second metaheuristic that we studied is iterated local search (ILS), which is designed to overcome most of the issues of RLS, while retaining its simplicity. The idea behind ILS is to perturb the current locally optimal solution s^* to get a new center t and call again the local search procedure from there, obtaining a new local optimum t^*. If the new solution t meets an acceptance criterion, then t is chosen as the next starting point, otherwise it is rejected and the procedure is repeated from s^*. Intuitively, ILS implements a heuristic random walk on the set of locally optimal solutions of a given optimization problem. A high level pseudocode for ILS is given in Algorithm 1.

Algorithm 1: Basic ILS procedure

```
1 $s_0$ = GenerateRandomSolution ();
2 $s^*$ = LocalSearch ($s_0$);
3 repeat
4     $s'$ = Perturb ($s^*$, history);
5     $t$ = LocalSearch ($s'$);
6     $s^*$ = AcceptanceCriterion ($s^*$, $t$, history);
7 until termination condition ;
```

Note that the perturbation mechanism and the acceptance criterion are in general dependent on the history of the system: this allows for more effective and elaborate strategies. The simplest, yet very common, acceptance criterion is to accept the new solution t if and only if its objective value is better than that of s. Other strategies include a pure random walk option, in which the new solution t is always accepted, regardless of its cost, and a simulated annealing [8,11] like acceptance criterion based on temperature, in which t is always accepted if it is an improving solution, but is also accepted with a given probability even if its objective value is worse (the probability is usually dependent on the "temperature" of the system and on the difference between the two objective values, with slightly worsening steps being more likely). Note that the first two strategies do not make use of the history of the system, while the third does.

3 Exact Methods

Our approach to solve the problem to proven optimality is to formulate it as a (hopefully convex) mixed integer nonlinear program. While the description of the problem is highly nonlinear, it turns out that we can get rid of most nonlinearities (such as matrix inversions and conditional summations) by an appropriate extended formulation and off-line computations. In particular, given S as the set of samples, we can write the model as

$$\max \sum_s \sum_j \left[\log(\det(\overline{P}_{js})) + \log(\det(A_j))\right] / |S| \tag{1}$$

$$\sum_k u_{ik} = 1 \qquad \forall i \tag{2}$$

$$\overline{P}_{js} = A_j^{-1} + \sum_i \sum_k R_{ijks} u_{ik} \qquad \forall j \forall s \tag{3}$$

$$u_{ij} \in \{0,1\} \qquad \forall i \forall k \tag{4}$$

where

- $\overline{P}_{js}$ is the inverse of the posterior covariance matrix of target j in scenario s
- R_{ijks} is the inverse of the measurement covariance matrix between sensor i pointing in direction k and target j in scenario s if the target is within the FOV in this case, or the null matrix otherwise.
- u_{ik} is a binary variable whose value is 1 if and only if sensor i is pointing in direction k

Constraints (2) are typical assignment constraints, stating that each sensor must point in exactly one direction, while constraints (3) define the value of matrices $\overline{P}_{js}$. Note that in practice we cannot deal with matrix variables within most solvers, but this is easily taken care of because we are dealing with symmetric 2×2 matrices, and each of them can be encoded with 3 continuous free variables. Finally, note that the only nonlinearities left are the $\log\det(\cdot)$ terms in the objective function, and that, since $\log\det(\cdot)$ is concave in the positive semidefinite cone, the problem can formulated as a mixed integer convex program.

3.1 Generalized Benders decomposition

A closer look at the model in the previous section reveals a clearly decomposable structure: indeed, given an assignment of directions to sensors, the model splits into $|S| \times |J|$ subproblems, whose only role is to compute a piece of the (nonlinear) objective function. As such, the structure is amenable to a generalized form of Benders decomposition. In particular, we can introduce for each scenario s and target j an additional continuous variable θ_{sj}, representing the subexpression

$$\theta_{sj} = f(\overline{P}_{js}) = \log(\det(\overline{P}_{js})) + \log(\det(A_j))$$

and define a Benders subproblem as the (feasibility) problem:

$$\begin{cases} f(\overline{P}_{js}) \geq \theta^*_{sj} \\ \overline{P}_{js} = A_j^{-1} + \sum_i \sum_k R_{ijks} u^*_{ik} \end{cases}$$

where (u^*, θ^*) is any solution of the so-called master problem, namely

$$\begin{cases} \max \left[\sum_{s,j} \theta_{sj}\right] / |S| \\ \sum_k u_{ik} = 1 \\ \langle \text{Benders cuts} \rangle \\ u_{ik} \in \{0,1\} \\ \theta_{sj} \quad \text{free} \end{cases}$$

It is worth noting that, given a solution (u^*, θ^*) of the master, the value of the matrix $\overline{P}_{js}$ is uniquely defined. As such, in this case Benders cuts turn out to be simply outer approximation cuts of the nonlinear expression $f(\cdot)$, interpreted here as a function of three variables (those representing the corresponding symmetric 2×2 matrix). After some algebraic manipulation, and with a little abuse of notation, we get:

$$\theta_{sj} \leq f(\overline{P}^*_{js}) + \nabla f(\overline{P}^*_{js})(\overline{P}_{js} - \overline{P}^*_{js})$$

where

$$\overline{P}^*_{js} = A_j^{-1} + \sum_i \sum_k R_{ijks} u^*_{ik}$$

4 Computational Experiments

We implemented our codes in C++, using IBM ILOG CPLEX 12.5.1 [6] as black box MIP solver through the Cplex callable library APIs, and KNITRO 8.1 [13,3] as black box mixed integer nonlinear solver. All tests have been performed on a standard desktop machine, equipped with an Intel i7-2600 CPU running at 3.40GHz and with 16GB of RAM.

We tested our algorithms on five random instances, characterized as:

- the number of sensors m ranges from 4 to 8
- each sensor is placed at a random integer point in the plane
- the number of targets n is equal to 9
- the prior Gaussian distribution of each target is randomly chosen (the mean is again an integer point in the plane, while the covariance matrix is randomly chosen with entries in $[0, 1]$)
- the set of $K = 10$ directions is $\{0, \pi/5, \ldots, 9\pi/5\}$
- the FOV of each sensor is $\pi/5$
- the observation model H is taken as the identity matrix
- for each instance we sampled $S = 150$ scenarios to approximate the information gain expectation.

Instance characteristics are detailed in Table 1.

4.1 Landscape analysis

In order to evaluate the suitability of metaheuristic approaches to our directional sensor problem, we performed a preliminary search landscape analysis [12]. Intuitively, the search space of a combinatorial optimization problem can be thought of as a multidimensional landscape with hills, valleys and plateaus, and the performance of many metaheuristics algorithms is strongly influenced by its topology, such as the distribution of local minima and landscape ruggedness. Among the (many) techniques for computing synthetic indicators describing the properties of the search landscape, the fitness-distance correlation (FDC) [7] has proved to be a useful tool to predict the suitability of metaheuristic algorithms. Given a list L of $|L|$ feasible solutions for an instance of an optimization problem, each described by a pair (z_i, d_i), where z_i is the objective value of the solution and d_i is the distance to the closest global optimum, the FDC coefficient is simply defined as:

$$FCD(L) = \frac{c_{zd}}{s_z \cdot s_d}$$

where

$$c_{zd} = \frac{1}{|L|} \sum_{i=1}^{|L|} (z_i - \overline{z})(d_i - \overline{d})$$

$\overline{z}, \overline{d}$ are the average solution value and distance and s_z, s_d are the corresponding standard deviations. According to [7], values of the FDC coefficient greater than 0.15 (for a minimization problem), indicate a strong correlation between distance to optimum and objective value, suggesting that local-search based heuristics (among others) should perform well.

For each instance in our testbed, we used the RLS algorithm (with aggressive parameters) to sample a lot of feasible solutions and compute the FDC coefficients. Detailed results are shown in Table 1, while scatter plots for the bigger instances are given in Figure 1. Note that the distance function used is the L^1-norm of the difference between the two solution vectors, interpreted as vectors in $\mathbb{Z}^m$. According to the table, the FDC coefficient is always between 0.27 and 0.67, suggesting a good performance of metaheuristic algorithms.

Table 1 Instance characteristics.

instance	n	m	K	S	FDC
1	9	4	10	150	0.27
2	9	5	10	150	0.55
3	9	6	10	150	0.63
4	9	7	10	150	0.67
5	9	8	10	150	0.60

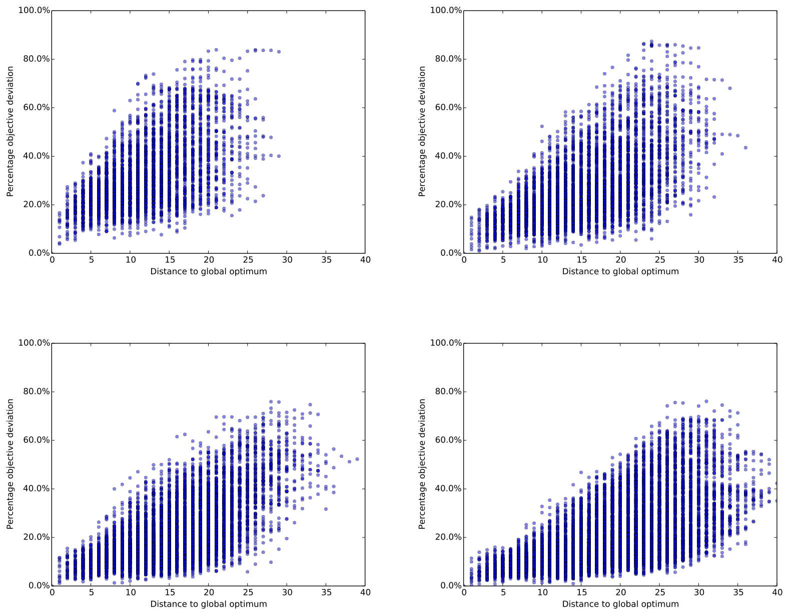

Fig. 1 Fitness-distance correlation plots for selected instances.

4.2 Metaheuristics results

We compared three basic heuristic approaches to our problem:

– a pure local search approach approach (`LS`)
– random restart local search (`RLS`), accepting only improving solutions. The method is given an iteration limit $L = 50$ and a no-improve limit of $NL = 10$.
– iterated local search (`ILS`), with iteration limit of $L = 50$. If no improvement is obtained in the last $NL = 10$ iterations, then the method is restarted from a new random solution.

All methods start from a randomly constructed solution.

In order to speed up computations on parallel architectures, we implemented a multi-threaded objective function evaluator. Such evaluation is trivially parallelizable, the contribution of each sample being independent of the others. We used 4 threads in our code, to match the number of available cores. Note that this is (positively) affecting all metaheuristics.

Table 2 reports average results for the instances in our testbed. Each algorithm was run 10 times starting from a different random solution. Median solution values and running times are reported. Column `opt.` reports the value of the optimal solution, as obtained by the exact algorithms of the next section. According to the table, both `RLS` and `ILS` are able to significantly improve upon the basic `LS` algorithm, while still being very fast on average (taking at most a few seconds on the largest instance). As far as the comparison between `RLS` and `ILS`, there is no clear winner, and both of them perform very satisfactorily, yielding the optimal solution in 4 out of 5 cases each.

4.3 Implementing Benders decomposition

Although the general Benders scheme is simple, there are quite a few design choices that must be made to come up with an efficient implementation. The first (and main) decision is how to implement the enumeration part. In particular, we have basically two options:

(a) [the original Benders method] keep the integrality requirement on the master variables, and solve an MIP to proven optimality using a black box MIP solver. If the optimal solution (u^*, θ^*) is not violated by any Benders cuts, then it is optimal for the original problem. Otherwise, add a few violated cuts to the master and repeat.

(b) [a modern branch-and-cut method] solve the master problem only once, but separate Benders cuts throughout the tree. In particular, integer feasible solution need always be checked.

Although common wisdom suggests that option (b) should be a superior implementation, in practice this is not always the case. In addition, even if option (b) eventually turns out to be faster than option (a), it may not be so for the whole duration of the solving process. Indeed, the two approaches have complementary strengths:

- option (a) can use the MIP solver as a black box, without the need to disable dual reductions and messing with callbacks (that can disable key features in some solvers). In addition, solving a MIP at each iteration has a powerful restart strategy built in. On the other hand, solving a MIP at each iteration can potentially waste a lot of effort, in particular at the end of the process, where masters tend to be quite hard.
- option (b) never wastes any enumeration effort (a single tree is maintained). However, at the very beginning the master formulation is usually a very poor approximation of the real model and the very first branching decisions (the most important ones) are not properly taken.

Table 2 Heuristics comparison.

	Time (s)			Objective			
instance	`LS`	`RLS`	`ILS`	opt.	`LS`	`RLS`	`ILS`
1	0.02	0.52	0.63	36.8854	36.1742	36.6552	36.8854
2	0.04	1.07	1.04	44.6929	43.4845	44.6929	44.6929
3	0.06	1.83	1.56	49.3872	48.0302	49.3872	49.3872
4	0.08	2.90	2.14	53.4560	51.8364	53.4560	53.4560
5	0.11	3.90	2.76	57.5305	56.3502	57.5305	57.3105

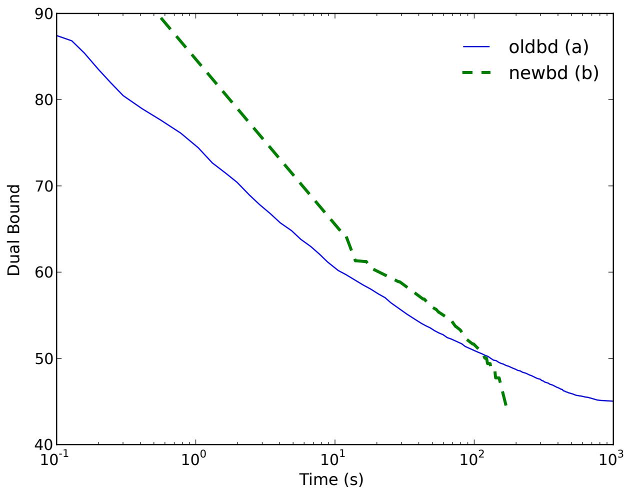

Fig. 2 Comparison of old-style and new-style Benders implementations.

A comparison between the two methods on an instance from our testbed is depicted in Figure 2.

While option (b) is clearly the winner in this case, the dual bound provided by option (a) is strictly better for the first 100 seconds of computation (approximately). The considerations above suggests that the branch-and-cut approach may benefit from a warm-start phase, provided by an old-style Benders implementation. Although such a simple idea is already quite an improvement (preliminary tests showed that this hybrid approach easily outperforms both options), it turns out that can we can devise an even more effective warm-start phase, by exploiting the metaheuristics presented in Section 2. In particular, we can use the randomized local search framework to sample many feasible solutions of our problem, generate outer approximation cuts from them, and add these cuts (as well as the best solution found) to the initial master formulation. In the following, we will denote by `BD` the straightforward implementation of option (b), and with `HBD` the version using this ad-hoc warm starting procedure.

4.4 Exact methods results

We compared two exact methods:

– a black box nonlinear solver (namely, KNITRO), on the MINLP model (1)-(4). We will refer to this method as `NLP`

– the generalized Benders approach described in Section 3.1, using a state of the art MIP solver (namely, IBM ILOG CPLEX) and its callback facilities. We provide two implementations of the methods, `BD` and `HBD`

Detailed results are given in Table 3. According to the table, all methods are able to solve to optimality the instances in our testbed, but with significantly different computing times. In particular, there is approximately a factor of 2-3 between `NLP` and `BD`. Note, however, that while CPLEX is a multi-threaded solver, KNITRO is not, so the difference between the two may not imply a ranking between the two methods, but rather between the two implementations. On the other hand, the improved Benders implementation, `HBD`, is one order of magnitude faster than `BD` (and thus also than `NLP`), and is the clear winner, solving the hardest instance in less the half an hour.

Table 3 Exact methods running times (in seconds).

instance	NLP	BD	HBD
1	139.32	22.85	6.12
2	409.96	110.20	9.78
3	1637.55	398.66	26.03
4	9188.71	3660.68	144.38
5	31619.70	20937.65	1757.69

5 Conclusions

The accomplishments of this work can be summarized are as follows:

– to transform the DSCP, non-convex in its original form, into a mixed-integer convex program
– to show that the landscape of the solution space is favorable to local search based metaheuristics, and show computationally that two of them can find consistently near optimal solutions in a matter of seconds
– to show that a black-box MINLP solver reliably solves the problem to optimality
– to develop a parallel Benders decomposition approach that, when hybridized and combined with the metaheuristic above, yields a very efficient solution procedure.

Even with a relatively large number of samples running time are in the range of minutes. In more complex and more general instances of the DSCP and related problems the ideas presented here should provide provably optimal solutions at a very reasonable computational effort.

Acknowledgements We thank Shankarachary Ragi and Edwin K. P. Chong for providing us the MATLAB code used to generate the instances in this paper and for the clarifications on the definition of the problem. The work of the first author was supported in part by AFOSR under grant FA9550-12-1-0153.

References

1. Ai, J., Abouzeid, A.A.: Coverage by directional sensors in randomly deployed wireless sensor networks. J. Comb. Optim **11**(1), 21–41 (2006)
2. Bertsekas, D.P., Tsitsiklis, J.N., Wu, C.: Rollout algorithms for combinatorial optimization. J. Heuristics **3**(3), 245–262 (1997)
3. Byrd, R., Nocedal, J., Waltz, R.: KNITRO: An integrated package for nonlinear optimization. In: G. di Pillo, M. Roma (eds.) Large-Scale Nonlinear Optimization, pp. 35–59. Springer-Verlag (2006)
4. Fusco, G., Gupta, H.: Selection and orientation of directional sensors for coverage maximization. In: SECON, pp. 1–9. IEEE (2009)
5. Güvensan, M.A., Yavuz, A.G.: On coverage issues in directional sensor networks: A survey. Ad Hoc Networks **9**(7), 1238–1255 (2011)
6. IBM: IBM ILOG Cplex Optimization Studio. http://www.cplex.com
7. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: L.J. Eshelman (ed.) Proceedings of the 6th International Conference on Genetic Algorithms, pp. 184–192 (1995)
8. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
9. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: Framework and applications. In: F. Glover, G. Kochenberger (eds.) Handbook of Metaheuristics, vol. 57, pp. 321–353. Kluwer Academic Publishers (2002)
10. Ragi, S., Mittelman, H.D., Chong, E.K.P.: Directional sensor control for maximizing information gain. In: Proceedings of SPIE conference ”Signal and Data Processing of Small Targets” (2013)
11. Černý, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications **45**(1), 41–51 (1985)
12. Weinberger, E.D.: Correlated and uncorrelated fitness landscapes and how to tell the difference. Biological Cybernetics **63**, 325–336 (1990)
13. Ziena: Ziena KNITRO. http://ziena.com/knitro.htm