A Survey of Meta-heuristics Used for Computing Maximin Latin Hypercube

Arpad Rimmel[1] and Fabien Teytaud[2]

[1] Supélec E3S, France
arpad.rimmel@supelec.fr
[2] Univ. Lille Nord de France, France
teytaud@lisic.univ-littoral.fr

Abstract. Finding maximin latin hypercube is a discrete optimization problem believed to be *NP*-hard. In this paper, we compare different meta-heuristics used to tackle this problem: genetic algorithm, simulated annealing and iterated local search. We also measure the importance of the choice of the mutation operator and the evaluation function. All the experiments are done using a fixed number of evaluations to allow future comparisons. Simulated annealing is the algorithm that performed the best. By using it, we obtained new highscores for a very large number of latin hypercubes.

1 Introduction

In order to realize a mathematical model of a complex system with D parameters, a fixed number s of experiments are made. The values of the parameters for each experiment can be seen as a point in a D-dimensional space. The choice of the s points is crucial because it will directly impact the quality of the model. A classical requirement for the points is to respect two criteria:

- they must be *evenly spread*. This guaranties a good exploration of the parameter space.
- they must be *non-collapsing*. This ensures the fact that even if a parameter is useless, two different experiments will not give the same result.

One way to choose the points while validating those criteria is to use maximin latin hypercube. A latin hypercube in dimension D and of size s is a set of s points in $[1..D]^s$ such that on each dimension, every point has a different value. Therefore, the points in a latin hypercube are *non-collapsing*. Latin hypercube have been introduced in [12]. An example is given on figure 1.

Let d_{min} be the minimal distance of all the distances between each pair of points. The maximin latin hypercube is the latin hypercube with the largest d_{min}. The points of a maximin latin hypercube are therefore *evenly spread*. The maximin criteria has been introduced in [9]. Several distance functions can be used. In this paper, we focus on the distance function l_2. To reduce clutter, we use the square of the distance.

The points that determine the minimal distance are called *critical points*.

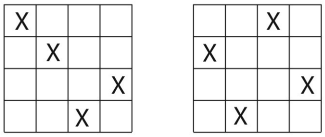

Fig. 1. Example of latin hypercubes of size 4 and dimension 2. left: $d_{min} = 2$. right: $d_{min} = 5$.

The complexity of finding a maximin latin hypercube depends on the distance function and on the dimension. For dimension 2 and both distance function l_1 and l_∞, a polynomial algorithm giving the maximin latin hypercube has been shown in [14]. For other dimensions and other distance functions, the complexity is not known but believed to be *NP*-hard.

Several meta-heuristics have been used in the literature to tackle this problem. The most classical ones are Genetic Algorithms (GA) [10], Simulated Annealing (SA) [13] and Iterated Local Search (ILS) [4]. Those three algorithms will be the focus of this paper and will be detailed later. We can also cite:

- Periodic Design (Perm) [1]. This algorithm uses periodic structures to generate latin hypercube. It gives good results for dimension 2 and correct results for dimension 3 but after that, the algorithm is outperformed by the others. For this reason and the fact that the principle of the algorithm is really different from the other, it will not be used in following comparisons.
- Enhanced Stochastic Evolutionary algorithm (ESE) [8]. This algorithm is based on a combination of GA and SA. Due to its similarity with SA, we do not use this algorithm in our experiments. However, we compare the highscores we find in this paper with the one obtained in [7].

In this paper, we present a comparison of the efficiency of GA, SA and ILS for a fixed number of evaluations. We first describe the principle of the algorithms. We then presents different mutation operators and several evaluation functions described in the literature. Finally, we give results on the efficiency of each algorithm with the different mutation operators and evaluation functions. The best combination allows us to obtain better highscores than those found in the literature for a large number of dimensions and sizes.

2 Algorithm Descriptions

In this section, we will briefly describe each algorithm. For further information, references are given in each section.

2.1 Genetic Algorithms

Genetic algorithms [5],[3] belong to the family of Evolutionary Algorithms.They are mainly used with a discrete search space, meaning they are used to address combinatorial optimization problems. Genetic Algorithms (GA) are techniques inspired by natural evolution, with in particular these following steps : (i) inheritance, (ii) mutation, (iii) crossover and (iv) selection. The principle is to have a population of candidate solutions evolving. Under the pressure selection, the population will converge toward better solutions. Algorithm 1 illustrates this method. λ denotes the population size, i.e. the number of individuals in the population.

Algorithm 1. Genetic Algorithm

```
Generate the initial population
Evaluate all individuals of the population
while the stopping condition not reached do
    for i from 1 to $\lambda$ do
        parent1 $\leftarrow$ parentalSelection(parentPopulation)
        if crossover probability is satisfied then
            parent2 $\leftarrow$ parentalSelection(parentPopulation)
            offspringPopulation[i] $\leftarrow$ crossover(parent1,parent2)
        else
            offspringPopulation[i] $\leftarrow$ parent1
        end if
        if mutation probability is satisfied then
            offspringPop[i] $\leftarrow$ mutate(offspringPop[i])
        end if
    end for
    Evaluate all individuals of offspringPopulation
    parentPopulation $\leftarrow$ survivalSelection(offspringPopulation, parentPopulation)
end while
```

2.2 Simulated Annealing

Simulated Annealing [2] is a global optimization algorithm, generally used with a discrete search space. The principle takes inspiration from the annealing principle in metallurgy. This technique consists in two phases : (i) heating and (ii) controlled cooling. In optimization, the method is to evolve a candidate solution thanks to a mutation. According to a certain probability a worse generated solution can be accepted. The idea behind this is to not get stuck in a local optimum. The controlled cooling represents this acceptance probability. Algorithm 2 represents this method.

2.3 Iterated Local Search

Iterated Local Search (ILS) [11] is based on a local search algorithm. An improvement over a single run of a local search is to launch it several times from a different starting point. A classical way of doing this is by choosing a new random starting point, the resulting algorithm is called the multistart approach

Algorithm 2. Simulated Annealing

```
Generate the initial solution $s$
Evaluate its fitness e ← f($s$)
$s_{best} \leftarrow s$
$e_{best} \leftarrow e$
while the stopping condition not reached do
    T ← temperature(iteration)
    $s_{new} \leftarrow$ mutate($s$)
    $e_{new} \leftarrow$ f($s_{new}$)
    if acceptance probability is satisfied or $e_{new}$ better than $e$ then
        $s \leftarrow s_{new}$
        $e \leftarrow e_{new}$
    end if
    if $e_{new}$ better than $e_{best}$ then
        $s_{best} \leftarrow s_{new}$
        $e_{best} \leftarrow e_{new}$
    end if
    increment iteration
end while
```

or random restart. The principle of ILS is to choose the new starting point by a perturbation of the last local minimum found instead.

The algorithm 3 is used for the local search. The initial solution is generated with a uniform distribution. The perturbation consists in a rotation of the values of several points on a given dimension.

In algorithm 3, the condition $*$ about the critical point is not necessary. Both possibilities will be studied in the following.

3 Mutations

In [6] 4 different neighborhoods are proposed. We have considered these 4 neighborhoods for our mutations and called them $m1$ to $m4$. The main principle of all these mutations is to change two points of the hypercube.

In the first one, the first point is chosen uniformly among all the critical points. The second point is chosen uniformly among all remaining points. For these two points, a random number of coordinates are changed.

In the second one, the two points are selected in the same way, but only one coordinate is changed. The coordinate is chosen randomly.

In the third one, the two points are selected in the same way and only one coordinate is changed. All the coordinates are tried and the one which gives the best d_{min} is selected. This modification add DIM evaluation to the algorithm, those evaluations are taken into account in the experiments.

In the last mutation, the two points are chosen uniformly among all points and the number of coordinates is randomly chosen.

Algorithm 3. Local Search of the ILS algorithm

```
$X = InitialSolution$
while there is an improvement do
  for i from 1 to size do
    for j from 1 to size do
      if i != j AND (i or j is a critical point)* then
        for k from 1 to DIM do
          $X' = X$
          in X' switch value of the points i and j on dimension k
          if $eval(X') > eval(X)$ then
            $X = X'$
          end if
        end for
      end if
    end for
  end for
end while
return X
```

4 Evaluation Functions

The natural evaluation function for a latin hypercube is the d_{min} function as this is the function used to determine if the latin hypercube is maximin. An other evaluation function ϕ_p has been proposed in [13]. It has the advantage of differentiating situations that would have a similar d_{min} value by using *all* the distances between points. This function is defined as follow:

– let $D_1(X), D_2(X), ..., D_R(X)$ be the set of all the distances between two points of X ordered in increasing order. R is the number of different distances in X.
– let $J_i(X)$ be the number of occurrences of $D_i(X)$.

$$\phi_p(X) = \left(\sum_{r=1}^{R} \frac{J_r(X)}{(D_r(X))^p}\right)^{\frac{1}{p}}, \; p \text{ is a parameter.}$$

5 Experiments

In this section, we make several experiments with the most efficient algorithms for the latin hypercube problem: *Genetic Algorithm (GA)*, *Simulated Annealing (SA)* and *Iterated Local Search (ILS)*. We first determine which value to use for each algorithm's parameter. Then, we evaluate the effect of the choice of the mutation function. Finally, we determine the effect of the evaluation function.

For the comparison between the algorithms to be fair and not implementation-dependant, we chose to fix the number of time the evaluation of a latin hypercube has to be computed. Unless it is stated otherwise, the number of evaluations used for the experiment is $100000*DIM$. We checked that the execution time depends only on the number of evaluations.

As testing every combination of dimension and size would be to time consuming, we chose arbitrarily three set of values that will be used for the experiments: (DIM 4 / SIZE 25), (DIM 9 / SIZE 10) and (DIM 8 / SIZE 20).

5.1 Effect of Algorithm Parameters

In this section, we experiment with different parameter values for each algorithm in order to determine which parameter has a high influence on the performance of the algorithm. It will also be used to fix the parameters for the next sections. For those experiments, the mutation operator $m3$ and the evaluation ϕ_{20} are used.

Genetic Algorithm. The main parameters of the genetic algorithm are λ and Gen.

Table 1. Effect of the modification of the ratio λ/Gen for GA

LatinHyperCube	$\lambda 100, Gen1000$	$\lambda 200, Gen500$	$\lambda 500, Gen200$
4/25	158.2±0.3	162.1 ± 0.2	155.9 ± 0.2
9/10	151.3±0.2	153.0 ± 0.1	153.9 ± 0.1
8/20	411.8±0.3	413.0 ± 0.2	401.6 ± 0.2

We see in table 1 that the parameters have a moderate impact on the performance. The ratio λ/Gen that gives the best performance is 2/5. This ratio will be used in the rest of the paper.

Table 2. Performance of SA for different initial and final probabilities

(DIM/SIZE)	0.5/0.1	0.3/0.1	0.5/0.01	0.3/0.01	0.1/0.01	0.05/0.01
4/25	162.64±0.1	171.97±0.1	170.24±0.09	175.24±0.07	175.87±0.08	174.9±0.1
9/10	154.33±0.02	155.35±0.02	155.15±0.02	155.34±0.02	154.86±0.04	154.07±0.07
8/20	418.71±0.1	427.88±0.1	429.64±0.09	434.02±0.08	435.33±0.07	434.84±0.08

Simulated Annealing. We see in table 2 that the value of the initial and final probabilities are very important parameters for the performance of the algorithm. We chose 0.3 for the starting probability and 0.01 for the final probability in the following experiments.

On the contrary, we see in table 3 that the ratio between the number of cycles and the number of trials is insignificant for the performance. We use a ratio of 10/1 for the rest of this paper.

Table 3. Comparison with mutation3 and eval_phi with simulated annealing

LatinHyperCube (DIM/SIZE)	# cycles	# of trials per cycles	avg scores
4/25	1000	100	175.24±0.07
	500	200	175.31± 0.07
	200	500	175.32±0.07
	100	1000	175.2±0.07
9/10	1000	100	155.34±0.02
	500	200	155.34±0.02
	200	500	155.3±0.02
	100	1000	155.34±0.02
8/20	1000	100	434.02±0.08
	500	200	434.96±0.07
	200	500	433.89±0.07
	100	1000	434.08±0.07

Iterated Local Search. Iterated Local Search does not have many parameters. We will however compare different version of the algorithm.

– v1: in the local search, all the points are considered
– v2: in the local search, only the critical points are considered.
– v3: multistart version of the algorithm v1.

Table 4. Performance of ILS for different versions of the algorithm

LatinHyperCube	ISL v1	ILS v2	ILS v3
4/25	162.0±0.2	158.5± 0.3	159.7±0.2
9/10	153.3±0.1	152.7± 0.1	151.0±0.1
8/20	412.7±0.3	408.0± 0.3	409.5±0.2

The different version have similar performance with a statistically significant advantage for the v1 version. This version will be used in the following sections.

5.2 Effect of the Mutations

We compare the performance of the genetic algorithm and the simulated annealing algorithm with the different mutations described in section 2. The results are given in table 2.

The choice of the mutation operator has a huge impact on the performance of both algorithms. This seems like a good direction for future improvements. In the rest of the paper, we will use the mutation 3.

5.3 Effect of the Evaluation Function

In this section, we compare the performance of the different algorithms when optimizing using the d_{min} function of using the ϕ_p function. The score is always

Table 5. Comparison of the effect of different mutation operators on the performance of the algorithms

LatinHyperCube	mutation	GA	SA
4/25	m1	156.6±0.2	173.82±0.12
	m2	156.4±0.3	173.64±0.12
	m3	162.0±0.2	175.22±0.07
	m4	147.2±0.3	173.69±0.09
9/10	m1	149.7±0.2	155.77±0.02
	m2	152.0±0.1	155.88±0.02
	m3	152.9±0.1	155.34±0.02
	m4	147.7±0.2	155.68±0.02
8/20	m1	399.4±0.3	429.79±0.13
	m2	406.1±0.3	425.8±0.13
	m3	412.8±0.2	433.92±0.07
	m4	388.8±0.4	424.59±0.13

the d_{min} value, even when optimizing with ϕ_p. The results are presented in table 6.

Optimizing according to the ϕ_p function improves greatly the results for all 3 algorithms. The value of the parameter p seems to have not so much impact as long as it is greater than 1. In the following, we will use ϕ_{10} as our evaluation function.

5.4 Scalability of the Algorithms

The scalability of an algorithm represents the robustness of the algorithm in front of the number of evaluations. For instance, if an algorithm performs better with a large number of evaluations, it should be a good choice if the evaluations can be parallelized.

We launched each algorithm with its best configuration with different number of evaluations and measured the best d_{min} obtained. The results are given on figure 2.

SA performs better than the two other algorithms in every case. Furthermore, there is no algorithm that improves faster than the others when the number of simulations increases. So SA should stay the best even with larger number of simulations.

6 HighScores

A highscore is the best d_{min} value obtained for a particular $SIZE$ and DIM. It corresponds to the best known lower bond for the maximin value. The previous highscores were obtained with the algorithms Iterated Local Search (see [4]) and ESE (see [7]). According to our comparison, the simulated annealing algorithm with the mutation operator m3 and the evaluation function ϕ_{10} performs better. We launched this algorithm on latin hypercube of dimension 3 to 10 and of

Table 6. Comparison of the effect of different evaluation function on the performance of the algorithms

LatinHyperCube	Evaluation Function	GA	SA	ILS
4/25	d_{min}	142.4±0.3	155.75±0.18	118.2±0.4
	ϕ_1	153.6±0.2	166.07±0.06	149.9±0.3
	ϕ_5	167.4±0.2	176.47±0.06	165.7±0.2
	ϕ_{10}	165.8±0.2	176.47±0.06	165.5±0.2
	ϕ_{20}	161.8±0.2	175.26±0.07	162.0±0.2
9/10	d_{min}	144.5±0.2	150.00±0.10	138.3±0.2
	ϕ_1	153.3±0.1	155.58±0.04	153.8±0.1
	ϕ_5	153.4±0.1	155.68±0.02	153.9±0.1
	ϕ_{10}	153.5±0.1	155.64±0.02	153.7±0.1
	ϕ_{20}	153.0±0.1	155.29±0.02	153.3±0.1
8/20	d_{min}	381.5±0.4	406.18±0.21	332.6±0.7
	ϕ_1	396.5±0.3	415.16±0.11	376.2±0.5
	ϕ_5	410.4±0.2	430.68±0.09	404.1±0.3
	ϕ_{10}	414.9±0.2	434.29±0.07	413.1±0.2
	ϕ_{20}	413.2±0.2	434.06±0.08	412.7±0.2

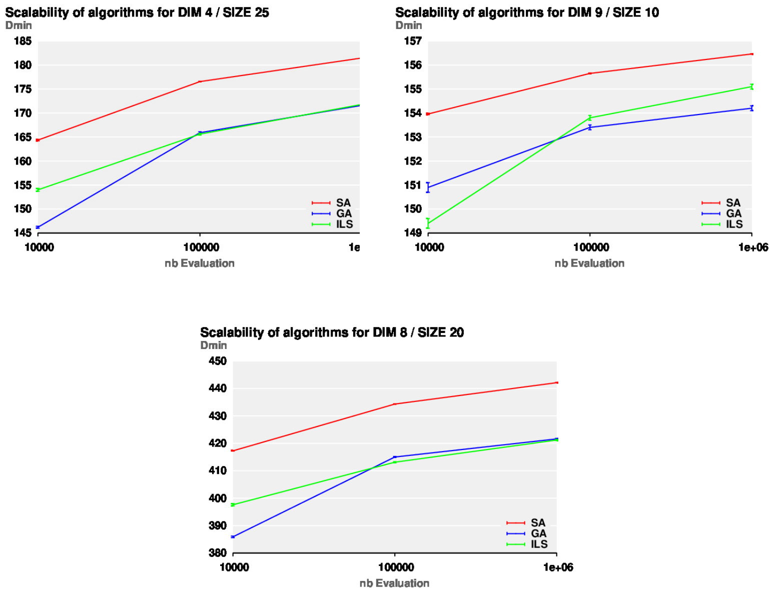

Fig. 2. Effect of the increase of the number of evaluations. We plot the minimal distance as a function of the number of evaluations. The higher the better.

size 2 to 25 with 10 000 000 evaluations. The tables 7 and 8 show the previous highscores obtained with ILS and ESE as well as the ones we get with SA. On a large majority of the latin hypercubes, we obtain similar or better result.

Table 7. Current highscores

SIZE	*DIM* = 3				*DIM* = 4			*DIM* = 5			*DIM* = 6		
	ILS	ESE	Per	SA	ILS	ESE	SA	ILS	ESE	SA	ILS	ESE	SA
2	3	3	3	**3**	4	4	**4**	5	5	**5**	6	6	**6**
3	6	6	3	**6**	7	7	**7**	8	8	**8**	12	12	**12**
4	6	6	6	**6**	12	12	**12**	14	14	**14**	20	20	**20**
5	11	11	6	**11**	15	15	**15**	24	24	**24**	27	27	**27**
6	14	14	14	**14**	22	22	**22**	32	32	**32**	40	40	**40**
7	17	17	14	**17**	28	28	**28**	40	40	**40**	52	52	**52**
8	21	21	21	**21**	42	42	**42**	50	50	**50**	66	63	**66**
9	22	22	21	**22**	42	42	**42**	61	61	**61**	82	75	**82**
10	27	27	21	**27**	50	47	**50**	82	82	**82**	93	91	**95**
11	30	30	24	**30**	55	55	**55**	80	80	**82**	110	108	**111**
12	36	36	30	**36**	63	63	**63**	91	91	**94**	140	136	**142**
13	41	41	35	**41**	70	70	**70**	102	103	**104**	139	138	**143**
14	42	42	35	**42**	77	77	**78**	116	114	**118**	156	154	**161**
15	48	45	42	**48**	89	87	**89**	131	129	**134**	173	171	**178**
16	50	50	42	**50**	92	93	**94**	152	151	**154**	192	190	**196**
17	53	53	42	**56**	99	99	**102**	159	158	**163**	212	208	**221**
18	57	56	50	**57**	108	108	**114**	170	170	**174**	238	231	**247**
19	62	59	57	**62**	118	119	**122**	187	184	**191**	259	256	**266**
20	66	65	57	**66**	137	130	**137**	206	206	**208**	284	279	**291**
21	69	68	65	**69**	147	145	**149**	229	223	**232**	306	302	**314**
22	76	72	69	**82**	147	150	**153**	241	235	**244**	331	325	**347**
23	76	75	72	**82**	155	159	**164**	250	250	**260**	351	348	**364**
24	78	81	76	**83**	170	170	**171**	265	266	**274**	378	374	**388**
25	84	86	91	89	174	178	**183**	282	285	**294**	408	400	**419**

Table 8. Current highscores

SIZE	*DIM* = 7			*DIM* = 8			*DIM* = 9			*DIM* = 10		
	ILS	ESE	SA	ILS	ESE	SA	ILS	ESE	SA	ILS	ESE	SA
2	7	7	**7**	8	8	**8**	9	9	**9**	10	10	**10**
3	13	13	**13**	14	14	**14**	18	18	**18**	19	19	**19**
4	21	21	**21**	26	26	**26**	28	28	**28**	33	33	**33**
5	32	32	**32**	40	40	**40**	43	43	**43**	50	50	**50**
6	47	47	**47**	54	53	**54**	61	61	**62**	68	68	**68**
7	62	61	**62**	71	70	**72**	80	80	**81**	90	89	**91**
8	79	79	**80**	91	90	**91**	102	101	**103**	114	114	**116**
9	95	92	**95**	114	112	**114**	128	126	**128**	143	142	**144**
10	112	109	**113**	133	131	**133**	157	154	**158**	173	171	**175**
11	131	129	**132**	153	152	**157**	181	178	**184**	209	206	**211**
12	155	152	**158**	181	177	**183**	208	204	**213**	240	235	**243**
13	181	178	**184**	208	205	**213**	242	235	**246**	274	268	**279**
14	219	215	**220**	242	236	**245**	275	268	**282**	312	305	**318**
15	222	220	**228**	277	273	**280**	315	309	**321**	356	347	**363**
16	249	241	**253**	326	317	325	357	352	**364**	403	393	**411**
17	269	266	**278**	331	332	**343**	404	396	**413**	451	442	**462**
18	297	291	**304**	367	361	**374**	466	451	**469**	505	496	**515**
19	323	323	**335**	398	390	**408**	466	469	**491**	569	554	**576**
20	356	349	**369**	426	425	**446**	512	506	**528**	640	625	**645**
21	386	380	**397**	467	463	**482**	550	548	**570**	647	650	**672**
22	421	418	**433**	505	501	**525**	593	595	**623**	697	691	**720**
23	452	448	**466**	545	542	**566**	649	640	**667**	736	747	**774**
24	486	481	**506**	591	585	**609**	694	690	**718**	805	800	**837**
25	525	520	**536**	631	626	**657**	746	739	**769**	866	857	**893**

7 Conclusion

We presented three classical algorithms to find maximin latin hypercube: genetic algorithm, iterated local search and simulated annealing. The experiments were done with a fixed number of evaluations to allow easier future comparisons. We show that the choice of the mutation operator has a great impact on the performance of each algorithm. Using a surrogate evaluation function also significantly improve the results. We then compared the 3 algorithms and the way they scale up and find that simulated annealing performs better than the other. Finally, we used simulated annealing on longer time settings to obtain scores as good as possible for different latin hypercube. This allowed us to obtain new highscores on most of them.

References

1. Bates, S.J., Sienz, J., Toropov, V.V.: Formulation of the optimal latin hypercube design of experiments using a permutation genetic algorithm. AIAA 2011, 1–7 (2004)

2. Bohachevsky, I.O., Johnson, M.E., Stein, M.L.: Generalized simulated annealing for function optimization. Technometrics 28(3), 209–217 (1986)
3. Goldberg, D.E., Holland, J.H.: Genetic algorithms and machine learning. Machine Learning 3(2), 95–99 (1988)
4. Grosso, A., Jamali, A., Locatelli, M.: Finding maximin latin hypercube designs by iterated local search heuristics. European Journal of Operational Research 197(2), 541–547 (2009)
5. Holland, J.H.: Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U. Michigan Press (1975)
6. Husslage, B., Rennen, G., Van Dam, E.R., Den Hertog, D.: Space-filling Latin hypercube designs for computer experiments. Tilburg University (2006)
7. Husslage, B.G., Rennen, G., van Dam, E.R., den Hertog, D.: Space-filling latin hypercube designs for computer experiments. Optimization and Engineering 12(4), 611–630 (2011)
8. Jin, R., Chen, W., Sudjianto, A.: An efficient algorithm for constructing optimal design of computer experiments. Journal of Statistical Planning and Inference 134(1), 268–287 (2005)
9. Johnson, M.E., Moore, L.M., Ylvisaker, D.: Minimax and maximin distance designs. Journal of Statistical Planning and Inference 26(2), 131–148 (1990)
10. Liefvendahl, M., Stocki, R.: A study on algorithms for optimization of latin hypercubes. Journal of Statistical Planning and Inference 136(9), 3231–3247 (2006)
11. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. International series in operations research and management science, pp. 321–354 (2003)
12. McKay, M.D., Beckman, R.J., Conover, W.J.: Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2), 239–245 (1979)
13. Morris, M.D., Mitchell, T.J.: Exploratory designs for computational experiments. Journal of Statistical Planning and Inference 43(3), 381–402 (1995)
14. Van Dam, E.R., Husslage, B., Den Hertog, D., Melissen, H.: Maximin latin hypercube designs in two dimensions. Operations Research 55(1), 158–169 (2007)