Department of Mathematics and Computer Science
University of Southern Denmark, Odense

February 20, 2017
Marco Chiarandini

DM841 - Heuristics and Constraint Programming for Discrete Optimization

Reexam, Fall 2016

Deadline: 15th March 2017 at 12:00.

- This is the reexam in DM841. It will be graded with external censorship.
- The assignment has to be carried out individually.
- The submission is electronic via Blackboard.
- You have to hand in:
  - the source code of your implementation of a CP and heuristic solver. Submit all your files in a `.tgz` archive. Your code will be compiled and run, hence it must comply to the requirements listed in this document.
  - A report that describes the work you have done and presents the results obtained. The document should not exceed 15 pages and must be in PDF format. You cannot list source code. You can write in Danish or in English.
- Changes to this document after its first publication on February 20 may occur. They will be emphasized in color and if they are major they will be announced via BlackBoard.
- Read all this document before you start to work.

---

1 The Problem

A *board* consists of $N \times N$ cells arranged in an $N \times N$ grid. Each cell is connected to at most eight neighbors along the four axes: vertical, horizontal and two diagonal. A *wrap-around board* has connections also between the left-most and right-most columns and between the top and bottom rows.
For two strings s and r of K bits $b_1 b_2 \dots b_K$ with $b_i \in \{0, 1\}$ the Hamming distance $d(s, r)$ is the number of bits at which the two strings differ.
Given a wrap-around board $N \times N$ and an integer $D \in \mathbb{Z}^+$, we wish to find an assignment of distinct strings all made of $K \geq \lceil 2 \log_2 N \rceil$ binary bits to each cell of the board in such a way that the maximal Hamming distance between any pair of neighboring cells is smaller than or equal to D. In the optimization sense, given a wrap-around board $N \times N$ we wish to determine the smallest D for which such assignment exists.
For example, the board 4×4 with bit strings of length $K = \lceil 2 \log_2(4) \rceil = 4$ admits a solution for $D = 2$ (see Figure 1) but it does not admit one for $D = 1$. Hence, $D = 2$ is the best possible maximal Hamming distance for this case. The binary strings can be represented by integer numbers in base-10 in which case we wish to assign to each of the N^2 cells a distinct integer from $\mathbb{Z}_{2^K}$.
In the following we will identify an instance of the problem by the triple (N, K, D).

```
1111 0101 1100 0110          15   5  12   6
0111 1101 0100 1110           7  13   4  14
1011 0001 1000 0010          11   1   8   2
0011 1001 0000 1010           3   9   0  10
```

Figure 1: An assignment with $D = 2$ for the 4×4 board. On the left the binary strings disposed on the board; on the right the representation of the strings in base-10.

2 Background

The NK-model problem defined above was motivated by the following application. A current focus in management science is the investigation of human decision-making when searching for new alternatives. Experimental work in this area uses the *NK* model of rugged performance landscapes [**?**].
The *NK* model defines a combinatorial search space, consisting of all possible (binary) string of length N. For each string in this search space, a scalar value, called the *fitness*, is defined. If a distance metric is also defined between strings, the resulting structure is a landscape. Fitness values are defined according to a specific model, but the key feature of the *NK* model is that the fitness of a given string s is the sum of contributions from each bit (locus) b_i in the string:

$$F(s) = \sum_{i=1}^{N} f(b_i)$$

and the contribution from each bit in general depends on the bit itself and the value of K other bits:

$$f(b_i) = f(b_i, b_{i_1}, \ldots, b_{i_K})$$

where b_{i_j} are the other bits upon which the fitness of b_i depends. Hence, the fitness function is a mapping between strings of length $K+1$ and scalar values. (Source: `http://en.wikipedia.org/wiki/NK_model`.) An example of NK landscapes at varying K is given in Figure 2.
In experiments within management science, human subjects are asked to search for high-performing product configurations. They combine N attributes — the N parameter in the *NK* model — to specify a product configuration. The value or payoff of a particular configuration is initially unknown and is only discovered after trying out the configuration. The complexity of alternatives — the K parameter — is captured by allowing for interactions among the attributes in the payoff function. As complexity increases and interactions among attributes proliferate, the problem of finding a high-performing configuration becomes more challenging and difficult. The subjects in the experiments have only a limited number of search trials, far fewer than the number of all possible configurations. The aim of the experiments is then studying the opportunity cost of searching a new alternative varying the experimental treatment of the complexity K of the performance landscape.
A dominant search strategy detected in the experiments is *neighborhood search*: the move to another position occurs by changing from 1 to 3 attributes. More specifically, for $N = 10$, experiments showed that out of 7539 active searches, 75.9% were within distance 3 from the current position with an average distance of 2.53, 84.5% within distance 4 and 90.3% within distance 5. In other terms, humans do not change radically the attributes available but perform local changes. Other observations show the importance of performance feedback [**?**].

Given these results there is interest to design an experiment in which the distance factor is removed. In other terms, researchers in management science would like to have an *NK* model in which the user is given the possibility to only move to new search states that are in the neighborhood of the current one. A way to achieve this is to provide to the humans an easy graphical representation of the possible moves available. A chess board in which moves are only allowed between neighboring cells seems well suited for this task. The problem is then to dispose the 2^K different search alternatives in the board in such a way that the maximum distance between any pair of adjacent cells is minimized.

3 Known facts

Focusing on the case $K = \lceil 2 \log_2 N \rceil$, where the length of the strings is just the minimal needed to have a different string for every cells of the board, Søren Haagerup, a former student of this course, has shown that:

1. the problem can be reformulated in terms of graphs as the problem of finding an embedding f of the quadri-axial torus G in the Hamming graph $H(K)$ with restricted dilation D. Since H has no clique of size 3 but there are several cliques of size 3 in G then there exist no solution with $D = 1$ for any N.

2. there is an easy method to construct solutions with $D = 2$ for all N that are powers of 2, ie, $N = 2^i$, for $i = 1, 2, 3, ...$

3. for any $N > 1$ with $2\lceil \log_2(N) \rceil = \lceil 2 \log_2(N) \rceil$, one can construct a solution with $D = 2$ if N is even and with $D = 4$ if N is odd. Table 1 makes explicit these results for N up to 30.

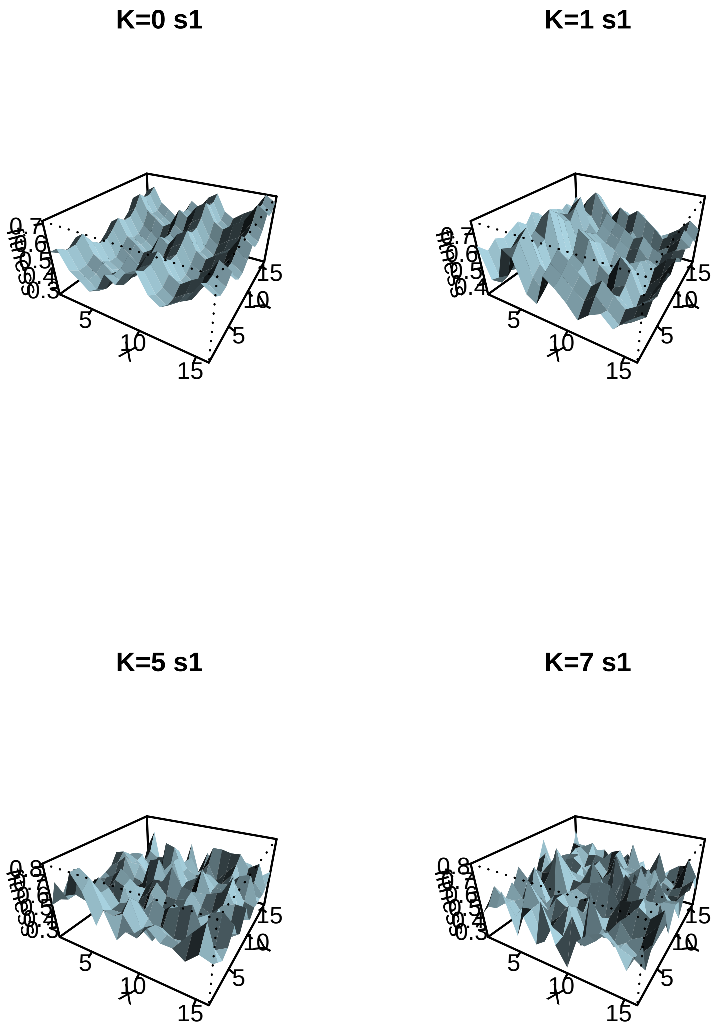

Figure 2: Different NK landscapes.

4 Open problems

- We do not know if the theoretical lower and upper bounds could get tighter for odd N and $2\lceil \log_2(N) \rceil = \lceil 2\log_2(N) \rceil$. For example: Does there exist an odd N, with solutions for $D = 2$? The first case without answer up to now is $(N, K, D) = (7, 6, 2)$.
- For all N where $2\lceil \log_2(N) \rceil \neq \lceil 2\log_2(N) \rceil$ can we get some upper bounds?
- Table 2 reports the best impossibility (lower bound) and possibility (upper bound) results for D when $K = \lceil 2\log(N) \rceil$. An assignment with the best known D has been found either by constructive method or by computational methods such as integer programming and heuristics. Instances with an asterisk are closed, that is, the best known D is also optimal.

5 Project Requirements

The aim of the project is to study CP and heuristic algorithms for solving the *decision version* of the problem described above. An instance of the problem is fully determined simply by the triple (N, K, D). Since very

N	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$2\lceil\log_2(N)$	4	4	6	6	6	6	8	8	8	8	8	8	8	8	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
$\lceil 2\log_2(N)\rceil$	4	4	5	6	6	6	7	7	7	8	8	8	8	8	9	9	9	9	9	9	10	10	10	10	10	10	10	10	10
UB	2	4	–	4	2	4	–	–	–	4	2	4	2	4	–	–	–	–	–	–	2	4	2	4	2	4	2	4	2

Table 1: Upper bounds obtained by construction for N=2,...,30.

N	K	Impossible	Possible	
2	2	1	2	*
3	4	3	4	*
4	4	1	2	*
5	5	2	3	*
6	6	1	2	*
7	6	1	3	
8	6	1	2	*
9	7	1	3	
10	7	1	3	
11	7	1	3	
12	8	1	2	*
13	8	1	3	
14	8	1	2	*
15	8	1	3	
16	8	1	4	
17	9	1	3	
18	9	1	3	
19	9	1	3	
20	9	1	3	

Table 2: Best known results in terms of D for instances with N from 2 to 20 and $K = \lceil 2\log(N)\rceil$

little is known about the hardness of theses instances we will initially focus on instances (N, K, D) with $K = \lceil 2\log(N)\rceil$, $N = \{2, \ldots, 20\}$, and $\{D = 2, \ldots, 10\}$. (If these instances turn out to be too easy you may increase N).

All the following points must be addressed to pass the exam:

1. Formulate the NK-model problem in constraint programming terms using finite domain integer variables. Focus on the *decision version* of the problem that asks whether there exists a solution if the maximum distance is fixed to D. Thus, an instance of the problem is fully determined by the triple (N, K, D). [Models using set variables are also possible and you are welcome to test them and report their results as a comparison but the primary goal here is to have a model with only integer variables.]
2. Implement the model in Gecode and report the essential parts of the code.
3. Experiment with different instances of the problem and different choices for the branching heuristics. Since very little is known about the hardness of the instances focus on instances (N, K, D) with $K = \lceil 2\log(N)\rceil$, $N = \{2, \ldots, 20\}$, and $\{D = 2, \ldots, 10\}$. (If these instances turn out to be too easy you may increase N).
4. List all the symmetries that you can recognize and handle them in your implementation either by adding constraints before the search to or by using a dynamic procedure.
5. Study the impact of random restarts.
6. Report your (best) results in a table like Table 3. In the cell write the time in seconds that your model needed to find a solution to the instance. Limit the run time of your algorithm to a maximum of 5 minutes for each (N, K, D) instance.[1]
7. Describe the work in the previous points in a report (use font size of 11pt and 3cm margins).
8. Design and implement stochastic local search or metaheuristic algorithms that comprise a construction heuristic and a local search.

[1]Times refer to machines in IMADA terminal room.

	1	2	3	4	5	6	7	8	9	10
1	–	–	–	–	–	–	–	–	–	–
2	–	–	–	–	–	–	–	–	–	–
3	–	–	–	–	–	–	–	–	–	–
4	–	–	–	–	–	–	–	–	–	–
5	–	–	–	–	–	–	–	–	–	–
6	–	–	–	–	–	–	–	–	–	–
7	–	–	–	–	–	–	–	–	–	–
8	–	–	–	–	–	–	–	–	–	–
9	–	–	–	–	–	–	–	–	–	–
10	–	–	–	–	–	–	–	–	–	–
11	–	–	–	–	–	–	–	–	–	–
12	–	–	–	–	–	–	–	–	–	–
13	–	–	–	–	–	–	–	–	–	–
14	–	–	–	–	–	–	–	–	–	–
15	–	–	–	–	–	–	–	–	–	–
16	–	–	–	–	–	–	–	–	–	–
17	–	–	–	–	–	–	–	–	–	–
18	–	–	–	–	–	–	–	–	–	–
19	–	–	–	–	–	–	–	–	–	–
20	–	–	–	–	–	–	–	–	–	–

Table 3: Example of table for presenting the final results: on the rows N and on the columns D.

9. Undertake an experimental analysis where you compare and configure the algorithms from the previous point.
10. Describe the work done in a report of at most 10 pages. The report must at least contain a description of the best algorithm designed and the experimental analysis conducted. The level of detail must be such that it makes it possible for the reader to reproduce your work.
11. Report the results of the best algorithms on the test instances made available in a table like Table 3.
12. Draw conclusions and compare CP and LS approaches.
13. Describe the work in the previous points in a report.

6 Remarks

Remark 1 For each point above a description must be provided in the report of the work undertaken. In particular for the best algorithm arising from the experimental analysis enough details must be provided in order to guarantee the reproducibility of the algorithm from the report alone (i.e., without need for looking at the source code). It is important to give account of the computational cost of the operations in the local search.

Remark 2 This is a list of factors that will be taken into account in the evaluation:

- quality of the final results;
- level of detail of the study;
- complexity and originality of the approaches chosen;
- organization of experiments that guarantees reproducibility of conclusions;
- clarity of the report;
- presence of the analysis of the computational costs involved in the main operations of the local search.
- effective use of graphics in the presentation of experimental results.

Remark 3 Note that a few, well thought algorithms are better than many naive ones!

Appendix A Solution checker

A program to verify your solutions is available at:

`http://www.imada.sdu.dk/~marco/DM841/Files/checker.cpp`

Your program must output the solution in a text file with one number per line. The number represents the binary string in decimal representation assigned to each cell of the board sorted by visited the board in row-wise order.
Compile the checker program with

```
g++ checker.cpp -o checker
```

and run the program with

```
checker -n 16 -k 8 -d 2 -c 16-8-2.sol
```

The file `16-8-2.sol` is also available at:

`http://www.imada.sdu.dk/~marco/DM841/Files/16-8-2.sol`

Appendix B Handing in Electronically

Your work must be handed in electronically via BlackBoard. The official receipt will be obtained at the BlackBoard submission.
This section describes how you must organize your electronic submission.
Main directory:

```
userID/
```

where `userID` is your login at SDU email

```
userID/README
userID/Makefile
userID/src/
userID/bin/
userID/res/
userID/doc/
```

The directory `doc` contains a pdf or postscript version of your report. The file README provides instructions for compilation of the program. The directory `src` contains the sources which may be in C, C++, Java or other languages. If needed a Makefile can be included either in the root directory or in `src`. After compilation the executable must be placed in `bin`. For java programs, a jar package can also be created via Makefile.
Programs must work on IMADA's computers under Linux environment and with the compilers and other applications present on IMADA's computers. Students are free to develop their program at home, but it is their own responsibility to transfer the program to IMADA's system and make the necessary adjustments such that it works at IMADA.[2]
The executable must be called `nk`. It must execute from command line by typing in the directory `userID/bin/nk`:

```
nk -n N -k K -d D -s SEED -c OUTPUT -t TIME
```

where beside the obvious flags to indicate the instance it is:

- `-t TIME` the time limit in seconds;
- `-s SEED` the random seed;
- `-c OUTPUT` the file name where the solution is written.

For example:

```
nk -n 4 -k 4 -d 3 -c 4-4-3.sol -t 120 -s 1 > 4-4-3.log
```

[2]Past issue: the java compiler path is `/usr/local/bin/javac`; in C, any routine that uses subroutines from the math.c library should be compiled with the `-lm` flag – eg, `cc floor.c -lm`.

will run the program on the instance $(4, 4, 3)$ for 120 seconds with random seed 1 and write the solution in the file `4-4-3.sol`.

In its default mode, the program must run the best algorithm developed and must print on the standard output 0 if the program never found a solution when it terminates and 1 if the program found a solution written in the solution file.

It is advisable to have a log of algorithm activities during the run. This can be achieved by printing further information on the standard error or in a file. A suggested format is to output a line whenever a new best solution is found containing at least the following pieces of information:

```
best 53 time 10.000000 iter 1000
```

All process times are the sum of user and system CPU time spent during the execution of a program as returned by the linux C library routine `getrusage`. If you are using the frameworks made available in the Graph Coloring Assignments continue to use the same functions for time checking.