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

October 9, 2008
Marco Chiarandini

DM811 - Heuristics for Combinatorial Optimization

Exam Project, Fall 2008

Note 1 The project is carried out individually and it is not allowed to collaborate. It consists of: algorithm design, implementation, experimentation and written report.

The evaluation of the project is based on the report. However, a program that implements the best algorithm described in the report must also be submitted. The program will serve to verify the correctness of the results presented. The report may be written in either English or Danish.

Note 2 Corrections or updates to the project description will be published on the course web page and will be announced by email to the addresses available in the Blackboard system. In any case, it remains students' responsibility to check for updates on the web page.

Note 3 *Submission*. An archive containing the electronic version of the written report and the source code of the program must be handed in through the Blackboard system **before 12:00 of Monday, 27 October 2007**. This is the procedure:

- choose the course DM811 in Blackboard,
- choose "Exam Project Hand in" in the menu on the left,
- fill the form and conclude with submit,
- print the receipt (there will be a receipt also per email).

See Appendix C for details on how to organize the electronic archive. Reports and codes handed in after the deadline will generally not be accepted. System failures, illness, etc. will not automatically give extra time.

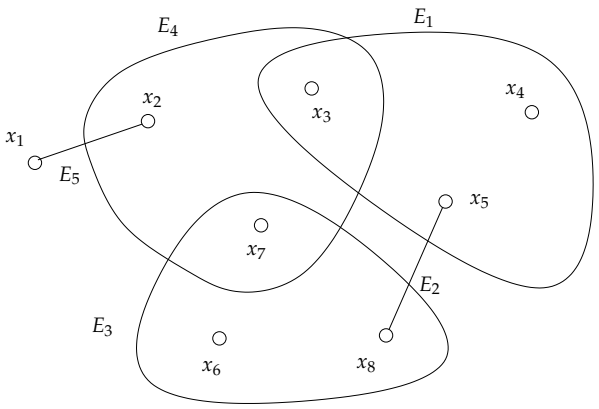
---

1 Problem Description

Given a finite ground set $X = \{x_1, x_2, \ldots, x_n\}$, a *hypergraph* is a collection $H = (E_1, E_2, \ldots, E_m)$ of distinct subsets of X. The elements of the set X are called *vertices*, and the elements of the set H are called *(hyper)edges*. We assume that $E_i \neq \emptyset$, for all $i = 1, 2, \ldots, m$, and $\bigcup_{i=1}^{m} E_i = X$. Moreover, we only deal with hypergraphs without loops, i.e., $|E_j| \geq 2$. A hypergraph H is called *r-uniform* if each edge $E \in H$ has exactly r elements. A 2-uniform hypergraph is a graph.

For a hypergraph H on X, a set $S \subseteq X$ is said to be *stable* (or *independent*) if it does not contain any edge E. A stable set is said to be *maximal* if it cannot be extended to a larger one. A stable set is said to be *maximum* if it includes as many vertices as possible. The *stability number* $\alpha(H)$ of H is the maximum cardinality of a stable set of H.

One can also define a hypergraph by its edge-vertex incidence matrix $A = [a_{ij}]$, with rows representing the edges $E_1, E_2, \ldots, E_m$ and columns representing the vertices $x_1, x_2, \ldots x_n$



$$A = \begin{array}{c} \\ E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 1: Representation of a hypergraph H and its incidence matrix. The set of vertices x_2, x_3, x_5, x_6 form a maximal stable set. The set of vertices x_1, x_4, x_5, x_6, x_7 is a maximal stable set that is also maximum.

and with $a_{ij} = 0$ if $x_j \notin E_i$ and $a_{ij} = 1$ if $x_j \in E_i$. See Figure 1 for an illustration of these concepts and [Ber89] for a theoretical treatment on hypergraphs.

The problem of the project is then defined as follows.

Definition. Maximum Stable Set Problem on Hypergraphs *(MSS)*
Input: *a hypergraph H on a ground set X.*
Task: *find a maximum stable set $S \subseteq X$ of H.*

If weights are associated to vertices then we may be interested in the stable set of maximum total weight. In its weighted version the Maximum Stable Set Problem on Hypergraphs is equivalent to the integer linear programming problem

$$\begin{array}{ll} \text{maximize} & c^{\mathrm{T}} x \\ \text{subject to} & Ax \leq p_A - \mathbf{1} \\ & x \in \{0,1\}^n \end{array}$$

where A is the edge-vertex incidence matrix of the hypergraph, the ith component of the vector p_A gives the number of ones in the row i of the matrix A, $\mathbf{1}$ is the vector of all ones of compatible dimension and c is a vector of nonnegative integer costs.

The generalization of the constraint $Ax \leq p_A - \mathbf{1}$ to $Ax \leq \mathbf{b}$, where $\mathbf{b}$ is a vector of nonnegative integer numbers with $b \leq p_A - \mathbf{1}$, leads to the Generalized Set Packing Problem which is defined as follows:

$$\begin{array}{ll} \text{maximize} & c^{\mathrm{T}} x \\ \text{subject to} & Ax \leq \mathbf{b} \\ & x \in \{0,1\}^n. \end{array}$$

The Maximum (Weighted) Stable Set Problem on Hypergraphs can be used for encoding constraint satisfaction problems (CSP) [PS03]. Assume we have a CSP with n variables $v_1, \ldots, v_n$ each with a domain d_i of size m_i. In its hypergraph representation (X, H) we have k variables with domain $\{0,1\}$, where $k = \sum_{i=1}^{n} m_i$. Variable $x_{ia} = 1$ corresponds to the instantiation $v_i = a$ in the CSP, and $x_{ia} = 0$ corresponds to $v_i \neq a$ in the CSP. The domain of each variable v_i in the CSP is represented by a clique K_m in (X, H), such that there are edges (z_{ia}, z_{ib}) for all $a, b \in d_i$. The constraints in the original CSP are represented as hyperedges in H. For example, a no-good $(v_i = a, v_j = b, \ldots, v_r = j)$ constraint in the

CSP has the corresponding hyperedge $(x_{ia}, x_{jb}, \ldots, x_{rj})$ in (X, H). Finding a solution to the CSP corresponds then to finding an stable set of size n in (X, H).

An example of application of the Generalized Set Packing Problem is in mechanism design for combinatorial auctions with single-minded bidders [Kry05]. A seller (auctioneer) wants to sell m kinds of goods U, to n potential customers (bidders). A good $e \in U$ is available in b_e units (supply). Suppose each bidder j can valuate subsets of goods: a valuation $v_j(S) \in \mathbf{R}$ for a subset $S \subseteq U$ means the maximum amount of money j is willing to pay for getting S. For simplicity, assume that bidders can bid for 0 or a single unit of a good, i.e., a_{ej} in the matrix A is in $\{0, 1\}$ for all $e \in U$ and $S \subseteq U$. An allocation of goods to bidders is a packing $S_1, S_2, \ldots, S_n \subseteq U$, i.e., bidder j gets set S_j and e appears at most b_e times in $S_1, S_2, \ldots, S_n$. The objective is to find an allocation with maximum social welfare, $\sum_j v_j(S_j)$. Each v_j is only known to bidder j and bidders are single-minded, i.e., for each bidder j there exists a set $S_j \subseteq U$ that he prefers and a $v_j^* \geq 0$, such that $v_j(S) = v_j^*$ if $S_j \subseteq S$ and $v_j(S) = 0$ otherwise.

For an application of GSP in scheduling see [Meg87].

2 Project Requirements

Preliminary tests conducted on the integer programming formulations solved by SCIP[1] showed that uniform instances of the MSS and unweighted instances of the GSP are harder to solve than non-uniform and weighted versions of these problems.

The aim of the project is, therefore, the study of heuristic algorithms to solve uniform and unweighted instances of the Maximum Stable Set Problem on Hypergraphs. A set of test instances is reported in Table 1. These instances are available for download from the course web site. The instances are large enough to require an efficient implementation of the algorithms designed. In the table, for some instances upper and lower bounds are provided. The name of the instance contains indication of its size, i.e, if `u-n-r-e-s.mss` then `n` is the number of vertices, `r` is the size of edges, `e` is the number of edges and `s` is the instance identifier.

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

1. Design and implement one or more construction heuristics.
2. Design and implement one or more local search algorithms.
3. Design and implement a high performing algorithm enhancing the heuristics at the previous two points with the use of stochastic local search methods.
4. For all the methods above carry out an experimental analysis and draw sound conclusions. For the algorithms at point 3 use 5 minutes as time limit per each single instance.[2]

3 Remarks

Remark 1 For each point above a description must be provided in the report of the work undertaken. In particular for the best algorithms arising from the experimental analysis enough details must be provided in order to guarantee the reproducibility of the algorithm from the report only (i.e., without having to look at the source code).

[1] `http://zibopt.zib.de/`

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

instance	upper bound	lower bound	gap
u-1000-10-1000-01.mss	910.49	792	13.01 %
u-1000-10-1000-02.mss	910.65	792	13.03 %
u-1000-10-1000-03.mss	910.54	791	13.13 %
u-1000-10-1000-04.mss	910.34	795	12.67 %
u-1000-10-1000-05.mss	910.28	790	13.21 %
u-1000-10-1000-06.mss	911.42	789	13.43 %
u-1000-10-1000-07.mss	910.30	792	13.00 %
u-1000-10-1000-08.mss	911.13	801	12.09 %
u-1000-10-1000-09.mss	909.84	793	12.84 %
u-1000-10-1000-10.mss	911.18	790	13.30 %
u-1000-50-1000-01.mss	981.01	924	5.81 %
u-1000-50-1000-02.mss	981.08	926	5.61 %
u-1000-50-1000-03.mss	981.01	928	5.40 %
u-1000-50-1000-04.mss	981.00	926	5.61 %
u-1000-50-1000-05.mss	981.09	927	5.51 %
u-1000-50-1000-06.mss	980.99	930	5.20 %
u-1000-50-1000-07.mss	980.88	931	5.09 %
u-1000-50-1000-08.mss	981.01	930	5.20 %
u-1000-50-1000-09.mss	981.01	927	5.51 %
u-1000-50-1000-10.mss	981.01	929	5.30 %
u-1000-10-10000-01.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-02.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-03.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-04.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-05.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-06.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-07.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-08.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-09.mss	1000.00	$-\infty$	infinite
u-1000-10-10000-10.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-01.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-02.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-03.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-04.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-05.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-06.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-07.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-08.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-09.mss	1000.00	$-\infty$	infinite
u-1000-50-10000-10.mss	1000.00	$-\infty$	infinite

Table 1: Four classes of random instances for the Maximum Stable Set Problem on Hypergraphs. The upper and lower bounds are produced by SCIP in 300 seconds of running time on an Intel Core 2 CPU 6300 at 1.86GHz, with 2048 KB cache and 2 GB RAM, running an Ubuntu 8.04 distribution of Linux with kernel 2.6.24-19-generic.

Remark 2 Besides the description of the algorithm, it will make a case for higher grade an analysis on the computational cost of the procedures implemented and details on the data structure used. The level of detail of the article [ARW08] might be taken as example for a high quality report.

Remark 3 The results of the experiments must be reported either in graphical form or in form of tables. Moreover, for the best solver resulting from the point 3, a table must be provided with the best results for each specific instance of Table 1.

Remark 4 The total length of the report should not be less than 8 pages and not be more than 14 pages, appendix included (lengths apply to font size of 11pt and 3cm margins). Although these bounds are not strict, their violation is highly discouraged. In the description of the algorithms, it is allowed (and encouraged) to use short algorithmic sketches in form of pseudo-code but not to include program codes.

Remark 5 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;
- originality of the experimental questions;
- organization of experiments which guarantee reproducibility and correctness of the conclusions;
- clarity of the report;
- effective use of graphics in the presentation of experimental results.

Appendix A Instance Format

The instance format is an extension of the DIMACS format used for several problems on graphs. Each line of the file begins with a letter that defines the rest of the line. The legal lines are:

- `c` comment: remainder of line ignored.
- `p type n m`, where `n` is the number of nodes (to be numbered $1, \ldots n$) and `m` the number of edges.
- `v u w`, where the first `v` is the letter indicating that the line is about a vertex, `u` is a number identifying the vertex (from 1 to n) and `w` is the weight of the vertex (all instances of the project have however weight 1).
- `e b` $\mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_k$ where `e` is the letter indicating that the line is about an edge, `b` is the size of the maximum number of vertices that can be picked from this edge (in the case of MSS instances this value is equal to $|E| - 1$ while for GSP it is equal to b_i in the model that defines the problem), $\mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_k$ are the vertices that compose the edge (k can be the same for all edges in the case of uniform hypergraphs or vary in the case of non-uniform hypergraphs).

Example for the instance of Figure 1:

```
p mss 8 5
v 1 1
v 2 1
v 3 1
v 4 1
v 5 1
v 6 1
v 7 1
v 8 1
e 1 1 2
e 2 2 3 7
e 2 3 4 5
e 2 7 6 8
e 1 8 5
```

Appendix B Solution Format

In order to check the validity of the results reported the program submitted must output the solution in a file when finishing. The format of the file is a column of numbers corresponding to the vertices selected to be member of the stable set. Vertices labels go from 1 to n. Each vertex must be preceded by the letter `s`.
Example:

```
s 1
s 4
s 5
s 6
s 7
```

A program to check the validity of the solution reported is made available at the course web page.

Appendix C Handing in Electronically

The electronic archive to hand in must be organized as follows. It expands in a main directory:

```
CPRN/
```

where CPRN is the student's first 6 digits of the CPR number (e.g., 030907) and its content:

```
CPRN/README
CPRN/Report/
CPRN/src/
```

The file README contains the manual for the 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 `src`. For java programs, a jar package can also be submitted.

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.[3]

The executable must be called mss. It will be run by typing in the directory CPRN/src/:

```
mss -i INSTANCE -t TIME -s SEED -o OUTPUT
```

- `-i INSTANCE` to load the data associated with the file INSTANCE.
- `-t TIME` to stop the program execution after TIME seconds. The test machine could not be totally dedicated at the moment of execution.
- `-s SEED` to initialize the random generator.
- `-o OUTPUT` the file name where the solution is written

For example:

```
mss -i u-100-50-1000-01.mss -o u-100-50-1000-01.sol -t 300 -s 1
```

will run the program on the instance `u-100-50-1000-01.mss` opportunely retrieved from the given path for 300 seconds with random seed 1 and write the solution in the file `u-100-50-1000-01.sol`.

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 853 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 library routine `getrusage`. Process times include the time to read the instance.

References

[ARW08] Diogo V. Andrade, Mauricio G. C. Resende, and Renato F. Werneck. Fast local search for the maximum independent set problem. Technical Report TD-7BBST2, AT&T Labs Research, Florham Park, NJ, USA, 2008. (available from `http://www.optimization-online.org/DB_HTML/2008/02/1898.html`).

[Ber89] Claude Berge. *Hypergraphs*. North-Holland, Amsterdam, Holland, 1989.

[Kry05] Piotr Krysta. Greedy approximation via duality for packing, combinatorial auctions and routing. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Proceedings of Mathematical Foundations of Computer Science, 30th International Symposium, MFCS 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 615–627. Springer, 2005. (Held in Gdansk, Poland, August 29 - September 2, 2005).

[3] 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`.

[Meg87] N. Megiddo. On the complexity of solving the generalized set packing problem approximately. Technical Report RJ 5898, IBM Almaden Research Center, San Jose, California, 1987.

[PS03] Patrick Prosser and Evgeny Selensky. A study of encodings of constraint satisfaction problems with 0/1 variables. In Barry O'Sullivan, editor, *International Workshop on Constraint Solving and Constraint Logic Programming*, volume 2627 of *Lecture Notes in Computer Science*, pages 121–131. Springer, 2003.