Chapter 1

Optimal technological architecture evolutions of Information Systems[*]

Vassilis Giakoumakis, Daniel Krob, Leo Liberti[†], and Fabio Roda

Abstract We discuss a problem arising in the strategic management of IT enterprises: that of replacing some existing services with new services without impairing operations. We formalize the problem by means of a Mathematical Programming formulation of the Mixed-Integer Nonlinear Programming class and show it can be solved to a satisfactory optimality approximation guarantee by means of existing off-the-shelf software tools.

1.1 Introduction

For any information system manager, a recurrent key challenge is to avoid creating more complexity within its existing information system through the numerous IT projects that are launched in order to respond to the needs of the business. Such an objective leads thus typically to the necessity of co-optimizing both creation *and* replacement/destruction — called usually *kills* in the IT language — of parts of the information system, and of prioritizing the IT responses to the business consequently.

This important question is well known in practice and quite often addressed in the IT literature, but basically only from an enterprise architecture or an IT tech-

Giakoumakis
MIS, Université d'Amiens, Amiens, France.
e-mail: `Vassilis.Giakoumakis@u-picardie.fr`

Krob, Liberti, Roda
LIX, École Polytechnique, 91128 Palaiseau, France.
e-mail: `{dk, liberti, roda}@lix.polytechnique.fr`

[*] Partially supported by École Polytechnique-Thales "Engineering of Complex Systems" Chair and ANR grant 07-JCJC-0151 "ARS".

[†] Corresponding author.

nical management perspective [2, 3, 10]. Architectural and managerial techniques, however, are often only parts of the puzzle that one has to solve to handle these optimization problems. On the basis of budget, resource and time constraints given by the enterprise management, architecture provides the business and IT structure of these problems. This is however not sufficient model them completely or solve them.

In this paper we move a step towards the integration of architectural business and IT project management aspects. We employ optimization techniques in order to model and numerically solve a part of this general problem. More precisely, we propose an operational model and a Mathematical Programming formulation expressing a generic global priorization problem occurring in the — limited, but practically rather important — context of a technological evolution of an information system (i.e. the replacement of an old IT technology by a new one, without any functional regression from the point of view of business). This approach promises to provide a valuable help for IT practitioners.

1.2 Operational model of an evolving information system

1.2.1 Elements of information system architecture

Any information system of an enterprise (consisting of a set D of departments) is classically described by two architectural layers:

- the *business layer*: the description of the business services (forming a set V) offered by the information system;
- the *IT layer*: the description of the IT modules (forming a set U) on which business services rely on.

In general, the relationship $A \subseteq V \times U$ between these two layers is not one-to-one. A given business service can require any number of IT modules to be delivered and vice-versa a given IT module can be involved in the delivery of several business services, as shown in Fig. 1.1.

1.2.2 Evolution of an information system architecture

From time to time, an information system may evolve in its entirety due the replacement of an existing software technology by a new one (e.g. passing from several independent/legacy software packages to an integrated one, migrating from an existing IT technology to a new one, and so on). These evolutions invariably have a strong impact at the IT layer level, where the existing IT modules $U^E = \{M_1, \ldots, M_n\}$ are replaced by new ones in a set $U^N = \{N_1, \ldots, N_{n'}\}$ (in the sequel, we assume $U = U^E \cup U^N$). This translates to a replacement of existing services (sometimes

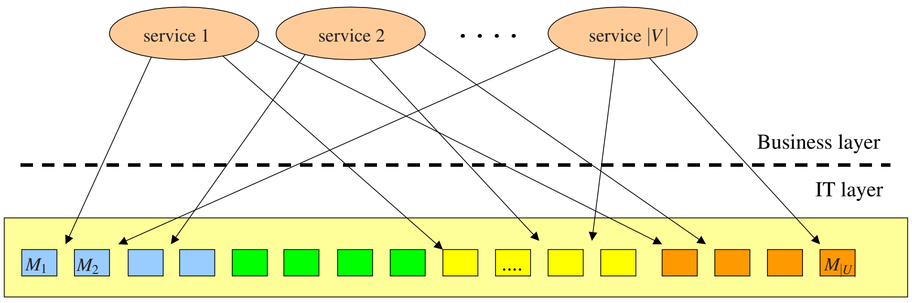

Fig. 1.1 A simple two-layer information system architecture.

denoted ES) in V by new services (sometimes denoted NS) in W ensuring that the impact on the whole enterprise is kept low, to avoid business discontinuity. This also induces a relation $B \subseteq W \times U^N$ expressing reliance of new services on IT modules. Note also that in this context, at the business level, there exists a relation (in $V \times W$) between existing services and new services which expresses the fact that a given existing service shall be replaced by a subset of new business services. We note in passing that this relation also induces another relation in $U^E \times U^N$ expressing the business covering of an existing IT module to a subset of new IT modules (see Fig. 1.2).

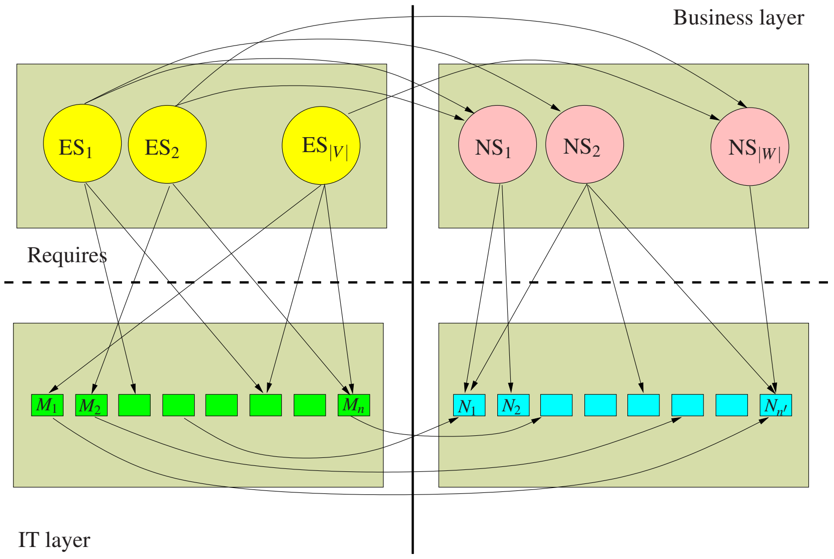

Fig. 1.2 Evolution of an information system architecture.

1.2.3 Management of information system architecture evolutions

Mapping the above information system architecture on the organization of a company, it appears clear that three main types of enterprise actors are naturally involved in the management of these technological evolutions which are described below.

1. *Business department managers*: they are responsible of creating business value — within the perimeter of a business department in the set D — through the new business services. This value might be measured by the amount of money they are ready to invest in the creation of these services (business services are usually bought internally by their users within the enterprise).
2. *IT project managers*: they are responsible for creating the new IT modules, which is a pre-requisite to creating the associated business services. The IT project manager has a project schedule usually organized in workpackages, each having a specific starting times and global budget (see Fig. 1.2).
3. *Kill managers*: they are responsible for destroying the old IT modules in order to avoid to duplicate the information system — and therefore its operating costs — when achieving its evolution. Kill managers have a budget for realizing such "kills", but they must ensure that any old IT module i is only killed after the new services replacing those old ones relying on i are put into service.

In this context, managing the technological evolution of an information system means being able of creating new IT modules within the time and budget constraints of the IT project manager in order to maximize both the IT modules business value brought by the new services and the associated kill value (i.e. the number of old services than can be killed).

1.2.4 The information system architecture evolution management problem

The architecture evolution of the IT system involves revenues, costs and schedules over a time horizon $t_{\max}$, as detailed below.

- *Time and budget constraints of the IT project manager*. Each new IT module $i \in U$ has a cost a_i and a production schedule.
- *IT module business value*. Each department $\ell \in D$ is willing to pay $q_{\ell k}$ monetary units for a new service $k \in W$ from a departmental production budget $H^\ell = \sum_{k:(\ell,k)\in F} q_{\ell k}$; the business value of the new service k is $c_k = \sum_{\ell:(\ell,k)\in F} q_{\ell k}$. We assume that this business value is transferred in a conservative way via the relation B to the IT modules. Thus, there is a business contribution β_{ik} over every $(i,k) \in B$ such that for each k we have $c_k = \sum_{(i,k)\in B} \beta_{ik}$; furthermore, the global business value of module i is $\sum_{k:(i,k)\in B} \beta_{ik}$. We also introduce a set $N \subseteq U$ of IT modules that are necessary to the new services.

- *Kill value*. Discontinuing (or *killing*) a module $i \in U$ has a cost b_i due to the requirement, prior to the kill, of an analysis of the interactions between the module and the rest of the system architecture, in order to minimize the chances of the kill causing unexpected system behaviour.

The evolution involves several stakeholders. The department heads want to maximize the value of the required new services. The module managers want to produce the modules according to an assigned schedule whilst maximizing the business value for the new services to be activated. The kill managers want to maximize the number of deactivated modules within a certain kill budget. Thus, the rational planning of this evolution requires the solution of an optimization problem with several constraints and criteria, which we shall discuss in the next session.

1.3 Mathematical Programming based approach

Mathematical Programming (MP) is a formal language used for modelling and solving optimization problems [12, 9]. Each problem is modelled by means of a list of index sets, a list of known parameters encoding the problem data (the *instance*), a list of decision variables, which will contain appropriate values after the optimization process has taken place, an objective function to be minimized or maximized, and a set of constraints. The objective and constraints are expressed in function of the decision variables and the parameters. The constraints might include integrality requirements on the decision variables. MPs are classified into Linear Programs (LP), Mixed-Integer Linear Programs (MILP), Nonlinear Programs (NLP), Mixed-Integer Nonlinear Programs (MINLP) according to the linearity of objective and constraints and to integrality requirements on the variables. MILPs and MINLPs are usually solved using a Branch-and-Bound (BB) method, explained at the beginning in Sect. 1.3.2. A *solution* is an assignment of numerical values to the decision variables. A solution is *feasible* if it satisfies the constraints. A feasible solution is *optimal* if it optimizes the objective function.

As explained above, an enterprise in our context consists of a set D of departments currently relying on existing services in V and wishing to evolve to new services in W within a time horizon $t_{\max}$. Each service relies on some IT module in U (the set $N \subseteq U$ indexes those IT modules that are necessary). The relations between services and modules and, respectively, departments and services, are denoted as follows: $A \subseteq V \times U$, $B \subseteq W \times U$, $E \subseteq D \times V$ and $F \subseteq D \times W$. If an IT module $i \in U$ is required by a new service, then it must be produced (or activated) at a certain cost a_i. When an IT module $i \in U$ is no longer used by any service it must be killed at a certain cost b_i. Departments can discontinue using their existing services only when all new services providing the functionalities have been activated; when this happens, the service (and the corresponding IT modules) can be killed. Departments have budgets dedicated to producing and killing IT modules, which must be sufficient to perform their evolution to the new services; for the purposes of this paper, we suppose that departmental budgets are interchangeable, i.e. all departments credit and debit their costs and revenues to two unique enterprise-level budgets: a production budget H_t and a kill budget K_t indexed by the time period t. A new service $k \in W$ has a value c_k, and an IT module $i \in U$ contributes β_{ik} to the value of the new service k that relies on it. We use the graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ shown in Fig. 1.3 to model departments, existing services, new services, IT modules and their relations. The vertices are $\mathscr{V} = U \cup V \cup W \cup D$, and the edges are $\mathscr{E} = A \cup B \cup E \cup F$. This graph is the union of the four bipartite graphs (U,V,A), (U,W,B), (D,V,E) and (D,W,F) encoding the respective relations. We remark that E and F collectively induce a relation between existing services and new services with a “replacement” semantics (an existing service can be killed if the related new services are active).

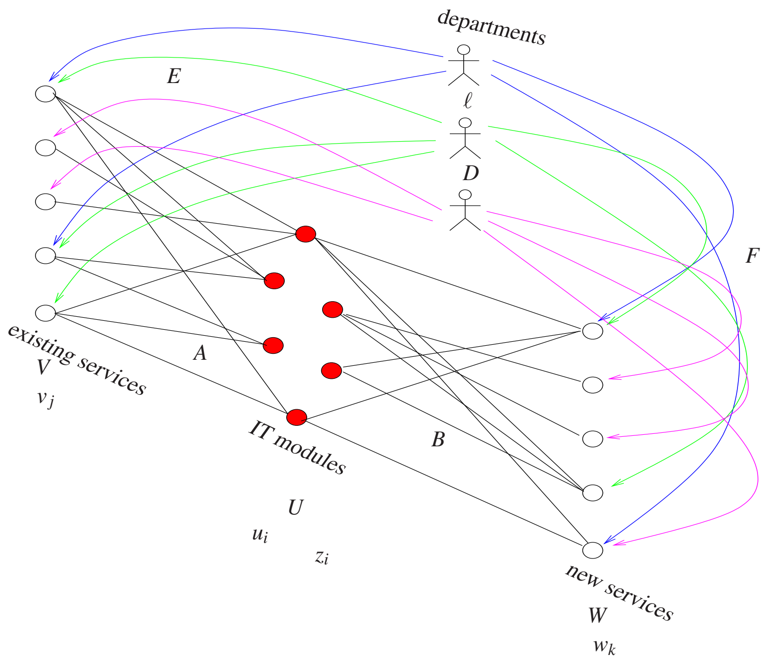

Fig. 1.3 The bipartite graphs used to model the problem.

Although in Sect. 1.3.1 we omit, for simplicity, to list explicit constraints for the production schedule of IT modules, these are not hard to formulate (see e.g. [4]), and do not change the computational complexity of the solution method we employ.

1.3.1 Sets, variables, objective, constraints

We present here the MP formulation of the evolution problem. We recall that NS stands for new service and ES for existing service.

1. **Sets**:
   - $T = \{0, \ldots, t_{\max}\}$: set of time periods (Sect. 1.2.4, p. 4);
   - U: set of IT modules (Sect. 1.2.1, p. 2);
   - $N \subseteq U$: set of IT modules that are necessary for the NS (Sect. 1.2.4, p. 4);
   - V: set of existing services (Sect. 1.2.1, p. 2);
   - W: set of new services (Sect. 1.2.2, p. 3);
   - $A \subseteq V \times U$: relations between ES and IT modules (Sect. 1.2.1, p. 2);
   - $B \subseteq W \times U$: relations between NS and IT modules (Sect. 1.2.2, p. 3);
   - D: set of departments (Sect. 1.2.1, p. 2);
   - $E \subseteq D \times V$: relations between departments and ES (Sect. 1.3, p. 6);
   - $F \subseteq D \times W$: relations between departments and NS (Sect. 1.3, p. 6).
2. **Parameters**:
   - $\forall i \in U \; a_i =$ cost of producing an IT module (Sect. 1.2.4, p. 1.2.4);
   - $\forall i \in U \; b_i =$ cost of killing an IT module (Sect. 1.2.4, p. 1.2.4);
   - $\forall t \in T \; H_t =$ production budget per time period (Sect. 1.3, p. 1.3);
   - $\forall t \in T \; K_t =$ kill budget per time period (Sect. 1.3, p. 1.3);
   - $\forall (i,k) \in B \; \beta_{ik} =$ monetary value given to NS k by IT module i (Sect. 1.2.4, p. 1.2.4).
3. **Decision variables**:

$$\forall i \in U, t \in T \; u_{it} = \begin{cases} 1 \text{ if IT module } i \text{ is used for a ES at time } t \\ 0 \text{ otherwise;} \end{cases} \tag{1.1}$$

$$\forall i \in U, t \in T \; z_{it} = \begin{cases} 1 \text{ if IT module } i \text{ is used for a NS at time } t \\ 0 \text{ otherwise;} \end{cases} \tag{1.2}$$

$$\forall j \in V, t \in T \; v_{jt} = \begin{cases} 1 \text{ if existing service } j \text{ is active at time } t \\ 0 \text{ otherwise;} \end{cases} \tag{1.3}$$

$$\forall k \in W, t \in T \; w_{kt} = \begin{cases} 1 \text{ if new service } k \text{ is active at time } t \\ 0 \text{ otherwise.} \end{cases} \tag{1.4}$$

4. **Objective function**. Business value contributed to new services by IT modules. This is part of the objective of the module managers, which agrees with the objective of the department heads.

$$\max_{u,v,w,y,z} \sum_{\substack{t \in T \\ (i,k) \in B}} \beta_{ik} z_{it} w_{kt}. \tag{1.5}$$

5. **Constraints**.
   - Production budget (cost of producing new IT modules; this is another objective of the module managers):

$$\forall t \in T \smallsetminus \{t_{\max}\} \quad \sum_{i \in U} a_i (z_{i,t+1} - z_{it}) \le H_t, \tag{1.6}$$

where the term $z_{i,t+1} - z_{it}$ is only ever 1 when a new service requires production of an IT module — we remark that the next constraints prevent the term from ever taking value -1.

- Once an IT module is activated, do not deactivate it.
$$\forall t \in T \smallsetminus \{t_{\max}\}, i \in U \quad z_{it} \le z_{i,t+1}. \tag{1.7}$$
- Kill budget (cost of killing IT modules; this is part of the objective of the kill managers):
$$\forall t \in T \smallsetminus \{t_{\max}\} \quad \sum_{i \in U} b_i (u_{it} - u_{i,t+1}) \le K_t, \tag{1.8}$$
where the term $u_{it} - u_{i,t+1}$ is only ever 1 when an IT module is killed — we remark that the next constraints prevent the term from ever taking value -1.
- Once an IT module is killed, cannot activate it again.
$$\forall t \in T \smallsetminus \{t_{\max}\}, i \in U \quad u_{it} \ge u_{i,t+1}. \tag{1.9}$$
- If an existing service is active, the necessary IT modules must also be active:
$$\forall t \in T, (i,j) \in A \quad u_{it} \ge v_{jt}. \tag{1.10}$$
- If a new service is active, the necessary IT modules must also be active:
$$\forall t \in T, (i,k) \in B : i \in N \quad z_{it} \ge w_{kt}. \tag{1.11}$$
- An existing service can be deactivated once all departments relying on it have already switched to new services; for this purpose, we define sets $\mathscr{W}_j = \{k \in W \mid \exists \ell \in D \, ((\ell, j) \in E \wedge (\ell, k) \in F)\}$ for all $j \in V$:
$$\forall t \in T, j \in V \quad \sum_{k \in \mathscr{W}_j} (1 - w_{kt}) \le |\mathscr{W}_j| v_{jt}. \tag{1.12}$$
- Boundary conditions. To be consistent with the objectives of the module and kill managers, we postulate that:
  - at $t = 0$ all IT modules needed by existing services are active, all IT modules needed by new services are inactive:
$$\forall i \in U \; u_{i0} = 1 \quad \wedge \quad z_{i0} = 0; \tag{1.13}$$
$$\forall j \in V \; v_{j0} = 1 \quad \wedge \quad \forall k \in W \; w_{k0} = 0. \tag{1.14}$$
  - at $t = t_{\max}$ all IT modules needed by the existing services have been killed:
$$\forall i \in U \quad u_{it_{\max}} = 0. \tag{1.15}$$

These boundary conditions are a simple implementation of the objectives of module and kill managers. Similar objectives can also be pursued by adjoining further constraints to the MP, such as for example that the number of IT modules serving ES must not exceed a given amount.

The formulation above belongs to the MINLP class, as a product of decision variables appears in the objective function and all variables are binary; more precisely, it is a Binary Quadratic Program (BQP). This BQP can either be solved directly using standard BB-based solvers [1, 11, 7] or reformulated exactly (see [8] for a formal definition of *reformulation*) to a MILP, by means of the PRODBIN reformulation [9, 5] prior to solving is with standard MILP solvers. A few preliminary experiments showed that the MILP reformulation yielded longer solution times compared to solving the BQP directly.

1.3.2 Valid cuts from implied properties

The BB method for for MPs with binary variables performs a binary tree-like recursive search. At every node, a lower bound to the optimal objective function value is computed by solving a continuous relaxation of the problem. If all integral variables happen to take integer values at the optimum of the relaxation, the node is *fathomed* with a feasible optimum. If this optimum has better objective function value than the feasible optima found previously, it replaces the *incumbent*, i.e. the best current optimum. Otherwise, a variable x_j taking fractional value $\bar{x}_j$ is selected for branching. Two subnodes of the current node are created by imposing constraints $x_j \le \lfloor \bar{x}_j \rfloor$ (left node) and $x_j \ge \lceil \bar{x}_j \rceil$ (right node) to the problem. If the relaxed objective function value at a node is worse than the current incumbent, the node is also fathomed. The step of BB which most deeply impacts its performance is the computation of the lower bound. To improve the relaxation quality, one often adjoins "redundant constraints" to the problem whenever their redundancy follows from the integrality constraints. Thus, such constraints will not be redundant with respect to the relaxation. An inequality is *valid* for a MP if it is satisfied by all its feasible points. If an inequality is valid for an MP but not for its relaxation, it is called a *valid cut*.

We shall now discuss two valid inequalities for the evolution problem. The first one stems from the following statement: *If a new service $k \in W$ is inactive, then all existing services linked to all departments relying on k must be active*. We formalize this statement by defining the sets:

$$\forall k \in W \quad \mathscr{V}_k = \{ j \in V \mid \exists \ell \in D\, ((\ell, j) \in E \wedge (\ell, k) \in F) \}.$$

The statement corresponds to the inequality:

$$\forall t \in T, k \in W \quad \sum_{j \in \mathscr{V}_k} (1 - v_{jt}) \le |\mathscr{V}_k| w_{kt}. \tag{1.16}$$

1.3.1 Lemma
Whenever (v,w) *are part of a feasible solution of the evolution problem,* (1.12) *implies* (1.16).

Proof. We proceed by contradiction: suppose (1.12) holds and (1.16) does not. Then there must be $t \in T, k \in W, j \in \mathscr{V}_k$ such that $w_{kt} = 0$ and $v_{jt} = 0$. By (1.12), $v_{jt} = 0$ implies $\forall h \in \mathscr{W}_j \ (w_{ht} = 1)$. By definition of $\mathscr{V}_k$ and $\mathscr{W}_j$, we have that $k \in \mathscr{W}_j$, and hence $w_{kt} = 1$ against the assumption. □

Thus, (1.16) is a valid inequality for the evolution problem.

The second inequality is a simple relation between v and w. First, we observe that the converse of Lemma 1.3.1 also holds; the proof is symmetric to that of Lemma 1.3.1: it suffices to swap j with k, $\mathscr{W}_j$ with $\mathscr{V}_k$, v with w, (1.12) with (1.16). Hence, (1.12) $\Leftrightarrow$ (1.16) for all feasible (v,w).

1.3.2 Proposition
The inequalities

$$\forall t \in T, j \in V, k \in W \ \exists \ell \in D \ ((\ell, j) \in E \wedge (\ell, k) \in F) \quad v_{jt} + w_{kt} \geq 1 \qquad (1.17)$$

are valid for the evolution problem.

Proof. Suppose (1.17) does not hold: hence there are $t \in T, j \in V, k \in W, \ell \in D$ with $(\ell, j) \in E$ and $(\ell, k) \in F$ such that $v_{jt} + w_{kt} = 0$. Since $v_{jt}, w_{kt} \geq 0$, this implies $v_{jt} = w_{kt} = 0$. It is easy to verify that if this is the case, (1.12) and (1.16) cannot both hold, contradicting (1.12) $\Leftrightarrow$ (1.16). □

Eq. (1.17) states that at any given time period no pair (ES, NS) related to a given department must be inactive (otherwise the department cannot be functional). We can add (1.16) and (1.17) to the MP formulation of the evolution problem, and hope they will improve the quality of the lower bound obtained via the LP relaxation. We remark that other valid inequalities similar to (1.16), (1.17) can be derived by the problem constraints; these will be studied in further works.

1.4 Computational Results

We aim to establish to what extent the proposed methodology can be used to solve realistically sized instances our problem. We first solve a set of small instances to guaranteed optimality and then a set of larger instances to within an approximation guarantee. We look at the CPU time and approximation guarantee behaviours in function of the instance size, and use these data to assess the suitability of the method.

We used the AMPL modelling environment [6] and the off-the-shelf CPLEX 10.1 solver [7] running on a 64-bit 2.1 GHz Intel Core2 CPU with 4GB RAM. Ordinarily CPLEX's Quadratic Programming (QP) solver requires QPs with Positive Semi-Definite (PSD) quadratic forms only. Although in our case this may not be true (depending on the values of β), CPLEX can reformulate the problem exactly to the required form because all variables are binary.

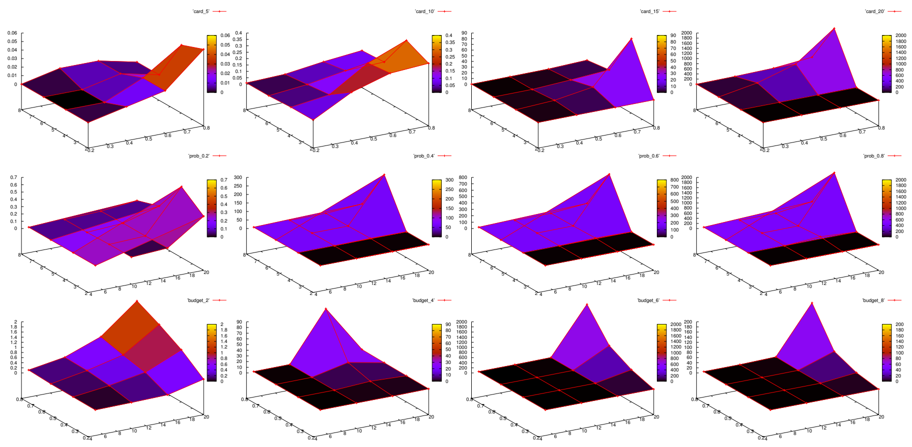

Fig. 1.4 CPU time when solving small instances to guaranteed optimality.

We consider a set of small instances, to be solved to guaranteed optimality, and one of larger instances where the BB algorithm is stopped either at BB termination or after 30 minutes of CPU time (whichever comes first). All instances have been randomly generated from a model that bears some similarity to data coming from an actual service industry. We consider three parameter categories: cardinalities (vertex set), graph density (edge creation probability) and monetary values. Each of the 64 instances in each set corresponds to a triplet (cardinality, edge creation probability, monetary value), each component of which ranges over a set of four elements.

In order to observe how CPU time scales when solving to guaranteed optimality, we present 12 plots referring to the small set, grouped by row. We plot seconds of user CPU time: for each fixed cardinality, in function of edge creation probability and monetary value (Fig. 1.4, first row); for each fixed edge creation probability, in function of cardinality and monetary value (Fig. 1.4, second row); for each fixed monetary value, in function of cardinality and edge creation probability (Fig. 1.4, third row). The largest "small instance" corresponds to the triplet $(20, 0.8, 8)$. The plots show that the proposed methodology can solve a small instance to guaranteed optimality within half an hour; it is also possible to notice that denser graphs and smaller budgets yield more difficult instances. Sudden drops in CPU time might correspond to infeasible instances, which are usually detected quite fast.

Fig. 1.5 is organized by rows as Fig. 1.4, but we plot the *optimality gap* — an approximation ratio — at termination rather than the CPU time, which is in this case limited to 30 minutes. The largest "large instance" corresponds to the triplet $(40, 0.8, 16)$. The optimality gap, expressed in percentage, is defined as $\left(\frac{100|f^* - \bar{f}|}{|f^* + 10^{-10}|}\right)\%$, where f^* is the objective function value of the best feasible solution found within the time limit, and $\bar{f}$ is the tightest overall lower bound. A gap of 0% corresponds to the instance being solved to optimality. The plots show that the proposed methodology is able to solve large instances to a gap of 14% within half an hour of CPU time at worst, and to an average gap of 1.18% within an average CPU time of 513s (just over 8 minutes).

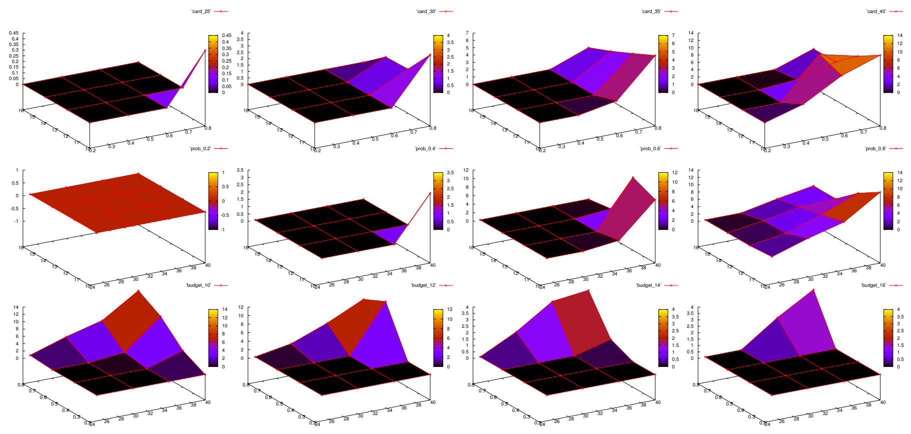

Fig. 1.5 Optimality gap when solving large instances within 30 minutes of CPU time.

It took 6h of user CPU time to reach a 15% gap in the real instance, which roughly corresponds to a triplet $(80, 0.2, 10)$. We managed, however, to reach a satisfactory 20% gap within 513s (the actual solution value improvement was $< 0.01\%$).

References

1. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software **24**(4), 597–634 (2009)
2. Bernus, P., Mertins, K., Schmidt, G.: Handbook on Architectures of Information Systems. Springer, Berlin (2006)
3. Caseau, Y.: Performance du système d'information – Analyse de la valeur, organisation et management (in French). Dunod, Paris (2007)
4. Davidović, T., Liberti, L., Maculan, N., Mladenović, N.: Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In: MISTA Proceedings (2007)
5. Fortet, R.: Applications de l'algèbre de Boole en recherche opérationelle. Revue Française de Recherche Opérationelle **4**, 17–26 (1960)
6. Fourer, R., Gay, D.: The AMPL Book. Duxbury Press, Pacific Grove (2002)
7. ILOG: ILOG CPLEX 10.0 User's Manual. ILOG S.A., Gentilly, France (2005)
8. Liberti, L.: Reformulations in mathematical programming: Definitions and systematics. RAIRO-RO **43**(1), 55–86 (2009)
9. Liberti, L., Cafieri, S., Tarissan, F.: Reformulations in mathematical programming: A computational approach. In: A. Abraham, A.E. Hassanien, P. Siarry, A. Engelbrecht (eds.) Foundations of Computational Intelligence Vol. 3, no. 203 in Studies in Computational Intelligence, pp. 153–234. Springer, Berlin (2009)
10. Luftman, J.: Competing in the Information Age. Oxford University Press, Oxford (2003)
11. Sahinidis, N., Tawarmalani, M.: BARON 7.2.5: Global Optimization of Mixed-Integer Nonlinear Programs, *User's Manual* (2005)
12. Williams, H.: Model Building in Mathematical Programming, 4th edn. Wiley, Chichester (1999)