UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea in Informatica

Realizzazione di uno strumento interattivo per il Metodo del Simplesso

Relatore: Prof.ssa **Enza Messina**
Correlatore: Dott.ssa **Ilaria Giordani**

Relazione della prova finale di:
Alessandro **PUTZU**
Matr. n. 067943

Anno Accademico 2008/2009

Indice

Indice delle figure

Introduzione

La **Ricerca Operativa (R.O.)**, dalla sua nascita come disciplina matematica applicata ad operazioni militari negli anni immediatamente precedenti alla Seconda Guerra Mondiale, ha portato a notevoli, e in alcuni casi superlativi, miglioramenti nell'efficienza, nel risparmio e nei guadagni di moltissimi settori dell'economia: industria di ogni genere, aziende che fanno uso di personale in numero elevato, logistica.

Qualsiasi realtà che si trovi ad affrontare problemi e decisioni, specialmente se riguardano l'allocazione di risorse limitate, può beneficiare dall'affrontarli nel *miglior modo* possibile e con il *minor costo*, o, che per molti versi è lo stesso, il *massimo ricavo*. Ognuno di questi è un problema di **ottimizzazione**, di cui si occupa l'omonima branca della **Ricerca Operativa.**

Sin dalla fine degli anni 60, nelle università furono incluse corsi di **R.O**. Uno degli argomenti principali di tali corsi è la **Programmazione Lineare (P.L.)**, che consiste nell'ottimizzazione di quantità esprimibili attraverso una funzione lineare e vincolate a risorse quantificabili con un sistema di equazioni e disequazioni lineari. Uno dei principali algoritmi iterativi per la risoluzione di problemi di P.L. è il **Metodo del Simplesso**, ideato dall'americano George Dantzig nel 1947.

Problemi con poche variabili e vincoli che non presentino casi particolari, possono essere facilmente risolti su carta, ma il controllo dei possibili errori commessi diventa presto una procedura tediosa e frequentemente soggetta ad errori.

Da ciò nasce l'esigenza di un nuovo strumento, che, automatizzando l'esecuzione del metodo e dei controlli sui dati inseriti, renda più facile lo svolgimento di esercizi di auto-apprendimento ed auto-valutazione, senza però escludere la necessità, per lo studente, di imparare **in maniera autonoma**. Il *tool*, quindi, dovrebbe svolgere le funzioni di esercitatore, ponendo domande allo studente e dando informazioni sulla correttezza o meno delle risposte.

Lo studio della letteratura ha rilevato la presenza di alcuni progetti analoghi: programmi o *applet* Java, spesso basati sul *web*, per la risoluzione di problemi di **Programmazione Lineare** di piccole dimensioni. Nella totalità delle soluzioni esaminate, però, si è notata la mancanza di alcuni requisiti necessari allo strumento voluto. In particolare, il requisito di interattività con lo studente era assente nella totalità dei *tool*. Difatti, la maggior parte dei programmi, una volta inseriti i dati, forniva la soluzione (o le soluzioni dei vari passi) con poca o nessuna partecipazione da parte dell'utente. Alcuni, però, offrivano funzionalità interessanti, che sono state studiate ed assimilate.

Lo scopo della presente tesi è sviluppare e introdurre un **nuovo strumento**, che risponda ai seguenti requisiti:

- **Interattività.** Scopo del programma non sarà *risolvere* un problema per lo studente, ma *porre domande* riguardanti la risoluzione del problema. Lo studente dovrà rispondere a queste domande; solo successivamente il programma confronterà la risposta data con quella elaborata internamente, dando l'esito ed una valutazione indicativa.

- **Specificità.** Il programma dovrà poter assistere nella risoluzione di *problemi lineari* di piccole dimensioni, da eseguire in un ambiente controllato (i.e. durante un'esercitazione in laboratorio informatico) oppure in autonomia. Non sarà progettato per problemi che eccedano le sue dimensioni massime, o che esulino dalla **Programmazione Lineare Continua**.

- **Semplicità d'uso.** Il programma sarà sviluppato con un'interfaccia che risulti al tempo stesso semplice e chiara. Sarà conforme alle più note linee guida **HCI** (*Human-Computer Interaction*).

- **Portabilità.** Il programma dovrà poter essere scaricabile da una piattaforma centralizzata, ridistribuibile ed utilizzabile sotto una varietà di ambienti, con una minima mole di requisiti.

Nel **primo** e nel **secondo capitolo**, verranno introdotti i metodi matematici utilizzati nel progetto: rispettivamente, la **Programmazione Lineare** e il **Metodo del Simplesso**.

Nel **terzo capitolo**, verrà esaminato lo **stato dell'arte** dei risolutori esistenti. Saranno presentati alcuni strumenti adatti alla risoluzione del **metodo del simplesso**, con le loro peculiarità, principali caratteristiche e mancanze rispetto allo scopo prefissato.

Nel **quarto capitolo** verrà descritto lo strumento sviluppato nel corso del progetto. Verranno spiegate nel dettaglio le metodologie e gli strumenti utilizzati durante lo sviluppo, la struttura, le principali scelte implementative e le funzionalità.

Infine, il **quinto capitolo** sarà dedicato all'analisi ed ai test effettuati sullo strumento, nonché alla presentazione dei risultati. Saranno qui presentati sia test di funzionalità in condizioni normali, con problemi tipici della didattica e di utilizzo previsto, che *stress test* con problemi di dimensioni massime o che presentano situazioni particolari, che possono essere o non essere presenti nel normale svolgimento delle lezioni ed esercitazioni; questi avranno lo scopo di visualizzare ed analizzare il comportamento del programma in condizioni critiche.

Capitolo 1

Cenni di Programmazione Lineare

Uno dei problemi più importanti della **Ricerca Operativa** consiste, dato un problema pratico e reale, nel trovare il **modello matematico** che meglio lo rappresenti.

Tre passi fondamentali sono richiesti per individuare un modello valido, qui di seguito riportati:

1. Esprimere il problema con una *funzione* matematica, che viene definita **funzione obiettivo**;

2. Definire i **vincoli** imposti da limitazioni di vario tipo nei problemi reali, quali disponibilità di tempo, risorse, tetti massimi di spesa e così via, espressi da equazioni o, più spesso, da disequazioni.

3. Trovare l'**ottimo** della **funzione obiettivo**, che è un punto (o *il* punto) di *massimo* o di *minimo*, a seconda del tipo di problema, nell'intervallo definito dai vincoli sopraimposti.

§1.1 – Definizione di Programmazione Lineare

Col termine **Programmazione Lineare (P.L.)**, si definisce quel ramo della Ricerca Operativa che si occupa di risolvere **problemi di ottimizzazione** in cui la **funzione obiettivo** e i **vincoli** siano funzioni lineari.

I primi problemi di Programmazione Lineare furono espressi da Leonid Kantorovich (URSS, 1912 – 1986) nel 1939 e, successivamente, la disciplina fu indipendentemente

sviluppata da George Dantzig (USA, 1914-2005). La **P.L.** raggiunse la forma attuale negli anni ’50 e, da allora, la sua diffusione crebbe costantemente. Sin dall'inizio, la Programmazione Lineare è stata utile per massimizzare i risultati, minimizzando i costi; un principio che trovò applicazione nell'industria bellica, aeronautica, elettronica, nella suddivisione del lavoro, nella ripartizione di materie prime e in innumerevoli altri campi. Ancora oggi, è uno strumento che permette alle imprese, indistintamente dalle più piccole alle più grandi, di risparmiare diverse migliaia o milioni di dollari che, altrimenti, sarebbero stati sprecati in prove o in soluzioni non ottimali. Si può così dedurre l’importanza che la **Programmazione Lineare** riveste, soprattutto nei problemi della realtà quotidiana.

Obiettivo di un problema di **P.L.** è trovare il **massimo** (o **minimo**) nella regione di spazio definita dai vincoli del problema. Questi possono rappresentare, come già accennato precedentemente, limiti di tempo, di spazio o quantità di materie prime o personale; ma anche, come ad esempio nei problemi su grafi (che pure sono rappresentabili come problemi di Programmazione Lineare), distanze tra due punti o tempi di percorrenza.

§1.2 – Algebra della Programmazione Lineare

La funzione obiettivo può essere formalizzata in questi termini:

$$Z = \sum_{i=1}^{n} c_i x_i$$

Dove:

- x_i è definita *variabile decisionale*;
- c_i è il *coefficiente* di tale variabile, che ne rappresenta l’*incremento* o *guadagno*;
- $\boldsymbol{Z}$ è il termine utilizzato per esprimere il valore della **funzione obiettivo**.

E’ possibile esprimere il prodotto, presentato nella formula appena proposta, in forma matriciale:

$$\boldsymbol{Z = c^T x}$$

Dove $\boldsymbol{c}^T$ e $\boldsymbol{x}$ indicano rispettivamente il vettore *colonna* dei coefficienti (trasposta del vettore dei coefficienti) e il vettore delle variabili decisionali.

I vincoli sono così espressi, come sistema lineare in forma matriciale:

$$\boldsymbol{Ax} \leq \boldsymbol{b}$$

In particolare, per ciascuno di essi si ha:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m$$

Dove i coefficienti $a_{11} \ldots a_{mn}$ rappresentano l'*incidenza* delle quantità x_i per ogni vincolo, e i parametri b_i sono i **termini noti** di tali vincoli, ovvero le quantità di *frontiera*.

Definiamo così la **forma standard [Hillier & Lieberman, 2005]** di un problema di programmazione lineare:

$$\text{Max. } \boldsymbol{Z} = \boldsymbol{c}^T\boldsymbol{x}$$

s.t.

$$\boldsymbol{Ax} \leq \boldsymbol{b}$$

$$x_{1\ldots n} \geq 0$$

dove **Max.** sta per *massimizzare* e **s.t.** sta per *Subject to*, "Soggetto a (i vincoli seguenti)". La condizione di non negatività si applica a tutti i modelli per cui non avrebbero senso valori negativi (ad esempio, se le variabili decisionali rappresentano quantità da allocare o impiegare: non si possono usare quantità *negative* di un qualche bene).

La **forma canonica** (anche detta **forma aumentata**) di un problema di programmazione lineare, invece, è definita come segue:

$$\text{Max. } \boldsymbol{Z} \text{ in}$$

$$\begin{bmatrix} 0 & \boldsymbol{c}^T & 0 \\ \mathbf{1} & \boldsymbol{A} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} Z \\ \boldsymbol{x} \\ \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{b} \end{bmatrix}$$

$$\boldsymbol{x} \geq \mathbf{0}, \boldsymbol{s} \geq \mathbf{0}$$

In quest'ultima, il problema è rappresentato come sistema di *equazioni* lineari, qui rappresentate sotto forma di moltiplicazione di matrici.

Il passaggio dalla **forma standard** alla **canonica**, è reso possibile dall'introduzione di un vettore di **variabili slack** (di **scarto**), rappresentate in $\boldsymbol{s}$ e il cui valore è inizialmente posto pari a 0 nella funzione obiettivo. Il numero di **variabili slack** è pari al numero dei vincoli. La loro funzione, così come l'utilizzo della forma aumentata, saranno spiegate in seguito; per ora, basti sapere che queste variabili rappresentano lo *scarto* tra le quantità effettive delle variabili in un vincolo, e il vincolo stesso. Anche queste variabili sono sottoposte a vincolo di *non-negatività*.

E' importante ricordare che, mentre il vettore $\boldsymbol{x}$ rappresenta le *variabili* del modello, i vettori $\boldsymbol{c}$, $\boldsymbol{b}$ e la matrice $\boldsymbol{A}$ ne sono i *parametri* noti, il cui valore deriva dalle scelte e misurazioni effettuate in fase di modellazione del problema. Perciò, la correttezza della soluzione deriva direttamente dalla correttezza del modello. Esistono metodi per valutare la correttezza *a posteriori*, quali l'**analisi di sensitività**.

§1.3 – Geometria della Programmazione Lineare

Si dice **soluzione** di un problema, una qualsiasi combinazione di valori del vettore $\boldsymbol{x}$.

Il sistema di disequazioni lineari dato dai vincoli, definisce un ente geometrico, nello spazio *n*-dimensionale, che nella geometria prende il nome di *politopo convesso*: l'intersezione convessa di più semispazi, analogo ad un poligono nello spazio bidimensionale e ad un poliedro nello spazio tridimensionale (e in verità, questi sono esempi di politopi). Nella **Programmazione Lineare**, il politopo così definito è chiamato **regione ammissibile**.

I punti che stanno all'interno di questa regione, cioè i punti che soddisfano *tutti* i vincoli, sono detti **soluzioni ammissibili**. Tra le soluzioni ammissibili, si avrà

sicuramente un valore che è il *migliore possibile*: il massimo (o minimo, a seconda dei casi) della funzione. Questa soluzione è detta **soluzione ottima**. La risoluzione di un problema di Programmazione Lineare, è la scoperta di *una* soluzione ottima.

Nella regione ammissibile, i punti all'intersezione di n vincoli (per un qualsiasi spazio n-dimensionale), prendono il nome di **vertici**. Riportiamo, senza dimostrare, la seguente affermazione **[Hillier & Lieberman, 2005]**:

> *Se un problema di programmazione lineare ammette soluzioni ammissibili e ha una regione ammissibile limitata, allora quel problema ha almeno una soluzione ottima ed almeno un vertice.*

E' evidente che, nel caso in cui *non* esistano soluzioni ammissibili, non può esistere nemmeno una soluzione ottima; allo stesso modo, se la regione ammissibile *non* è limitata, nemmeno la funzione Z può esserlo.

Il teorema di seguito riportato stabilisce una relazione tra la soluzione ottima e i vertici **[Archetti et al., 1989]**:

> *Il miglior vertice deve essere una soluzione ottima ed a sua volta, se la soluzione ottima è unica, essa deve essere un vertice. Se il problema ha più di una soluzione ottima, almeno due di queste devono essere vertici.*

Considerando il *significato* dei vincoli, ad esempio la disponibilità o l'allocazione di risorse, una conseguenza della precedente affermazione è che la **soluzione ottima** porterà a sfruttare almeno *n* dei vincoli, al massimo delle loro capacità.

A sua volta, la ricerca della soluzione ottima si riduce dalla possibilità di infiniti punti, almeno per quanto riguarda la **PL continua**, ad un numero *limitato* di punti: i vertici. Rimane da considerare che, tipicamente, i problemi di Programmazione Lineare possono avere migliaia di variabili e altrettanti vincoli, con combinazioni di *milioni* di vertici. Un algoritmo di ricerca "*brute force*", pertanto, non è applicabile nella maggior parte dei casi.

Di seguito verrà presentato un sistema efficiente e generico per calcolare la soluzione ottima di problemi di Programmazione Lineare, con il nome di **metodo del simplesso**.

Capitolo 2

Metodo del Simplesso

Il **metodo del simplesso** fu sviluppato e proposto nel 1947 dal sopracitato George Dantzig **[Hillier & Lieberman, 2005]**, nel corso dei suoi studi sulla Programmazione Lineare. In seguito, fu studiato e approfondito e incorporò alcuni miglioramenti; ancora oggi, la ricerca per incrementarne le possibilità e la velocità d'esecuzione procede.

Ebbe da subito un grandissimo successo, rendendo di fatto possibile (e non solo possibile, ma pressoché immediata) la risoluzione di problemi di Programmazione Lineare che avrebbero letteralmente richiesto *secoli* di calcoli, con possibili combinazioni in numero superiore agli atomi nell'intero Universo. Il continuo miglioramento degli strumenti (calcolatori elettronici) usati per l'implementazione dell'algoritmo, lo rende ancora più valido. Sebbene il **tempo di esecuzione** del simplesso non sia teoricamente polinomiale, lo è nella stragrande maggioranza dei casi **[Spielman & Teng, 2001]**.

§2.1 – Punti principali del Metodo del Simplesso

L'algoritmo, basandosi sulle precedenti affermazioni riguardanti i vertici della **regione ammissibile**, procede logicamente da un punto di partenza, alla soluzione ottima, spostandosi di volta in volta da un **vertice** ad uno *migliore*.

L'obiettivo del metodo del simplesso è identificare *un solo* vertice ottimo; è dimostrato che, data la proprietà di *convessità* della regione ammissibile, questo è sufficiente perché sia assicurata la presenza di un vertice ottimo **[Archetti et al, 1989]**. L'algoritmo si interrompe una volta raggiunto l'ottimo, oppure in presenza di condizioni che indicano l'*assenza* di un ottimo *finito*.

Il seguente pseudocodice permetterà di spiegare l'algoritmo in maniera più particolareggiata.

```
1:   imposta x a {0};
2:   while(!ottimo && !(impossibile || illimitato)) {
3:        if(x == ottimo)
4:             return x;
5:        porta(x_i)con max(c_i)in base;
6:        incrementa(x_i) al prossimo(v_j);
7:        per ogni(x_j) if(x_j != x_i)
8:             modifica(x_j) per rispettare i vincoli;
9:   }
```

La riga **1** contiene l'*inizializzazione* dell'algoritmo. Il metodo del simplesso necessita di un vertice iniziale; per questioni di facilitazione dei calcoli, *se possibile*, si seleziona l'origine.

L'iterazione parte alla riga **2**. Qui, con **`while(!`**`ottimo` **`&& !(`**`impossibile` **`||`** `illimitato`**`))`** si vuole sottolineare il fatto che l'algoritmo procede finché non trova una soluzione ottima, oppure finché diventa chiaro che sia impossibile o illimitato. Possono esistere, tuttavia, condizioni che impediscono di rilevare algebricamente l'impossibilità o l'illimitatezza del problema.

Le righe **3** e **4** contengono il passo di *uscita* dall'iterazione, il **test di ottimalità**. E' molto importante utilizzare un metodo appropriato per stabilire se un dato vertice, è ottimo oppure no. Visualizziamo il problema in due dimensioni (due **variabili decisionali**), come un grafico cartesiano in cui la regione ammissibile è un poligono. La *funzione obiettivo* è una retta, la cui intercetta rappresenta il valore di Z. Per ogni vertice, ci sono n vertici adiacenti, considerando il problema in uno spazio n-dimensionale; per ogni vertice che *non sia* ottimo, ce ne sarà **almeno uno** per cui, spostando la funzione obiettivo verso quel vertice, l'intercetta *aumenta*. Viceversa, se il vertice attuale è ottimo, spostando la funzione obiettivo verso *qualsiasi* vertice adiacente, il valore dell'intercetta *diminuisce*. Il test di ottimalità consiste quindi nel controllare se, per ogni vertice adiacente all'attuale, **non si produce** un aumento della funzione obiettivo

spostandosi verso una *qualsiasi* direzione.

Se il test di ottimalità ha successo, l'algoritmo termina e restituisce l'attuale vettore $\boldsymbol{x}$ delle variabili decisionali. Viceversa, si procede cercando la *direzione più vantaggiosa* lungo cui spostarsi. Questo si ottiene confrontando i valori dei **coefficienti** c_i per ogni variabile x_i; la variabile selezionata è quella con coefficiente c_i più alto, il che, passando dal modello al problema originario, significa scegliere di aumentare **l'attività che dà il maggior profitto**. Questo passo è rappresentato dalla riga **5** dello pseudocodice.

La riga successiva riporta un punto chiave dell'algoritmo del simplesso: l'incremento della variabile selezionata non è arbitrario, ma si arresta non appena incontra **l'intersezione** con un altro vincolo: in altre parole, uno dei vertici adiacenti. Il metodo algebrico per effettuare questa operazione assicura che l'incremento si arresti alla *prima* intersezione con un vincolo, il più restrittivo. L'ovvia motivazione è che, se questo vincolo è rispettato, allo stesso modo lo saranno tutti gli altri.

Le righe **7** e **8** rappresentano le cosiddette **operazioni di pivot**. La variabile modificata è il perno (o, per l'appunto, *pivot*) attorno a cui il problema viene riscritto, per mantenere valide le equazioni e disequazioni del sistema. Infine, l'algoritmo si ripete, con un nuovo **test di ottimalità** sulla soluzione appena calcolata.

§2.2 – *Nomenclatura e caratteristiche*

Per applicare il metodo del simplesso su un qualsiasi problema di programmazione lineare, è necessario porre il problema in **forma canonica**, come precedentemente specificato, ovvero esprimere i vincoli mediante *equazioni* (aumentate da **variabili slack** non-negative, una per ogni vincolo) e trasferire tutti i termini della funzione obiettivo al primo membro.

In un problema di programmazione lineare modellato con n variabili decisionali ed m vincoli, muoversi lungo i vertici della regione ammissibile significa, in termini matematici, eguagliare le quantità a *sinistra* di ogni vincolo, con il corrispondente termine noto. Per fare ciò, si pongono n variabili delle totali $m + n$ (di cui n sono le

variabili originali e le restanti, variabili **slack**) uguali a 0, ed m variabili (una per ogni vincolo) diverse da 0. Una tale combinazione prende il nome di **soluzione di base**. Una soluzione di base in cui tutte le variabili sono non-negative, si dice **soluzione di base ammissibile** (o *Basic Feasible Solution,* **BFS**).

Per ogni iterazione, le variabili poste pari a 0 sono dette **variabili non di base**; tutte le altre, **variabili di base.** Nel passaggio da un vertice ad uno adiacente, una variabile di base decresce fino a 0 (si dice che *esce dalla base*) e una variabile non di base, aumenta di valore (si dice che *entra in base*). La scelta delle variabili per questo scambio è effettuata secondo i criteri delle righe **5** e **6** dello pseudocodice, e i cui metodi algebrici saranno introdotti nel prossimo paragrafo. Si può facilmente vedere come due vertici adiacenti, abbiano le stesse variabili di base (pur con valori diversi), tranne una, e naturalmente lo stesso discorso vale per le variabili non di base.

La fase iniziale del metodo, necessita di una **BFS**: come è stato visto nella riga **1** dello pseudocodice, questa si ottiene ponendo (se possibile) tutte le variabili **decisionali** a 0. Si può facilmente notare che questo ha l'effetto di porre le variabili **slack** pari ognuna al termine noto del vincolo in cui compaiono. Lo spostamento da un vertice all'altro, si traduce nell'individuazione di successive **BFS**, ognuna corrispondente ad un vertice *migliore* del precedente.

§2.2.1 – Il tableau

Il **metodo del simplesso** richiede che il problema viene posto in una forma tabellare, detta **tableau**. Ponendo la funzione obiettivo nella prima riga e i termini noti a destra, alla prima iterazione il tableau avrà questa forma:

$$\begin{bmatrix} -\boldsymbol{c}_n & \boldsymbol{0}_m \\ \boldsymbol{A} & \boldsymbol{I}_m \end{bmatrix} \begin{bmatrix} Z \\ \boldsymbol{x} \\ \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{b} \end{bmatrix}$$

Dove:

- $-\boldsymbol{c}_n$ è il vettore, di n elementi, dei coefficienti delle variabili decisionali, nella funzione obiettivo, cambiato di segno perché le variabili sono state portate allo stesso membro di $\boldsymbol{Z}$;

- $\boldsymbol{A}$ è la matrice dei coefficienti di tutte le variabili decisionali nei vari vincoli, di dimensioni n per m, dove m è il numero di vincoli;
- $\boldsymbol{I_m}$ e $\boldsymbol{0_m}$ simboleggiano rispettivamente la matrice identità di ordine m e il vettore ad elementi nulli di dimensione m;
- $\boldsymbol{Z}$ è il valore della **funzione obiettivo**, con coefficiente costante pari a 1 nella funzione obiettivo, 0 in tutti i vincoli;
- $\boldsymbol{x}$ ed $\boldsymbol{s}$ rappresentano, rispettivamente, il vettore delle **variabili decisionali** e il vettore delle **variabili slack**;
- $\boldsymbol{b}$ è il vettore dei **termini noti** nei vincoli, di cardinalità m.

Come si può vedere, il **tableau** non è altro che la trasposizione in forma matriciale di un problema di Programmazione Lineare in **forma aumentata**.

Di seguito, riportiamo un esempio di *tableau* ottenuto da un problema con due variabili e due vincoli **[Hillier & Lieberman, 2005]**.

		x_1	x_2	s_1	s_2	
0)	Z	-3	-5	0	0	0
1)	s_1	1	0	1	0	4
2)	s_2	0	2	0	1	12

La forma tabellare permette di identificare in modo immediato la **BFS** corrente, il valore dell'ottimo e le condizioni di ottimalità e ammissibilità. In particolare, l'**ottimo** è il primo elemento nella colonna dei termini noti, precedentemente indicato con $\boldsymbol{Z}$. Nel *tableau* precedente, il valore dell'ottimo è 0. Nella **BFS**, le **variabili di base** hanno il valore riportato nella medesima colonna, mentre le **variabili non di base** hanno valore 0. Nella tabella appena descritta, la **BFS** ha valore (0, 0, 4, 12) con s_1 ed s_2 in base.

§2.3 – *Svolgimento del Metodo del Simplesso*

Durante l'elaborazione del metodo, si omettono la trascrizione delle varie x ed s e del segno di uguale, affiancando direttamente la colonna dei termini noti al resto del *tableau*. Si ha cura, inoltre, di segnare a lato quali siano le attuali **variabili di base**. Non è strettamente necessario, in quanto sono già ordinate, ma può essere una comodità in più segnare tutte le variabili sulla cornice superiore del *tableau.* Le varie colonne corrispondono ai coefficienti delle variabili (nella funzione obiettivo e nei vincoli).

Le variabili da scambiare di ruolo, una entrante in base e l'altra uscente, sono scelte mediante le **regole di pivot**. In particolare, per determinare quale variabile *entra* in base, si sceglie quella con il *minore* coefficiente c_i, che garantisce il *maggior* incremento del valore Z. Se due o più variabili presentano lo stesso valore di c_j, si potrebbe ottenere sempre la stessa sequenza di **BFS** e procedere all'infinito, senza mai arrivare all'ottimo. In casi come questo, si applica una delle regole cosiddette anti-ciclaggio: la più utilizzata (valida anche nel caso di due o più variabili candidate ad *uscire* di base) è la **regola di Bland**, che riportiamo qui senza dimostrazione.

> *Ogni volta che c'è più di una variabile candidata ad entrare in base si sceglie quella con indice più piccolo. Ogni volta che c'è più di una variabile candidata ad uscire dalla base, si sceglie quella con indice più piccolo.* **[Roma et al., 2008]**

Dopo aver scelto la variabile entrante, si stabilisce la variabile uscente attraverso il **test del minimo rapporto,** che costituisce la seconda regola di *pivot*.

Indichiamo con $\boldsymbol{A}_j$ la colonna corrispondente alla variabile entrante scelta e con a_{ij} l'elemento di $\boldsymbol{A}_j$ nella riga corrispondente al vincolo i-esimo. Se a_{ij} è maggiore di 0, si divide il termine noto b_i per questo valore; si ripete l'operazione per ogni elemento della colonna $\boldsymbol{A}_j$.

Il quoziente della divisione rappresenta la *massima* quantità di cui la variabile entrante possa aumentare, pur continuando a soddisfare il vincolo; aumentare una variabile di base a quella precisa quantità significa, a sua volta, muoversi lungo un vincolo (dato dalle altre variabili di base, che rimangono invariate, tranne l'uscente) fino all'intersezione con un altro.

Una volta effettuate le divisioni possibili, si sceglie come variabile uscente quella con il rapporto *minore* (da cui il nome del test) perché, altrimenti, la **BFS risultante** violerebbe uno o più vincoli. Questo, come precedentemente accennato, porta l'incremento ad arrestarsi alla *prima* intersezione con un vincolo.

Il passo successivo alla scelta delle **variabili, entrante** ed **uscente**, si effettuano le **operazioni di pivot** per riscrivere il problema in modo da tenere conto della nuova conformazione delle variabili (in altri termini, del nuovo vertice selezionato). Le operazioni da effettuare sono semplici manipolazioni algebriche sulle righe. Per prima cosa, si divide la riga i-esima per l'elemento di posto (i, j), detto **elemento pivot** (o **perno**, o **cardine**), in modo da ottenere un coefficiente pari a 1 per la nuova variabile di base, nel vincolo i-esimo. Successivamente, si somma la riga i-esima (già divisa) ad ogni altra riga del *tableau*, moltiplicata per l'inverso dell'elemento j-esimo (ovvero, l'elemento nella colonna *pivot*) di ogni riga. In questo modo, al termine dell'operazione, la colonna *pivot* verrà a contenere il valore 1 in corrispondenza della riga i-esima, e 0 in tutte le altre righe. Questa peculiarità del tableau si applica a tutte le **variabili di base** e fa sì che esse assumano (nella **BFS**) il valore del termine noto corrispondente b_i, perché tutte le altre variabili con coefficiente positivo sono **fuori base** e, analogamente, le altre **variabili di base** hanno coefficiente 0 in quella riga. Si può, a questo punto, riscrivere le variabili a lato del *tableau* segnando la nuova **variabile di base** al posto della variabile uscente.

Aggiornato il tableau, si ha una **nuova BFS**. Il problema, a questo punto, è risolto se *nessuno* dei coefficienti nella prima riga (eccettuato, naturalmente, il valore Z che non è un coefficiente) è *negativo*, il che corrisponde all'impossibilità di ottenere un miglioramento dell'ottimo *aumentando* una delle variabili non di base (si ricorda che, per la costruzione del *tableau*, si portano le variabili della funzione obiettivo allo stesso membro di $\boldsymbol{Z}$). Questo è il **test di ottimalità**. Se il problema non è stato risolto, si procede come prima, con l'applicazione delle condizioni di *pivot*, e così via.

Rimangono da testare le condizioni di *inammissibilità* e *illimitatezza* del problema. Un problema è inammissibile se *non esistono* soluzioni ammissibili per esso: i.e. se la **regione ammissibile** definita dai vincoli è vuota, ovvero, se i vincoli definiscono semispazi la cui intersezione è l'insieme vuoto. Nella **forma standard**, ovvero in un

problema di massimizzazione, con vincolo di non-negatività per tutte le variabili, soli vincoli di minoranza non stretta e termini noti tutti non negativi, un problema è sempre ammissibile (nel peggiore dei casi, la regione ammissibile si riduce ad un solo punto: l'origine). Nel caso di problemi in forma *non standard*, si adotta un procedimento diverso per l'implementazione del metodo del simplesso, detto **metodo del simplesso a due fasi**, di cui si parlerà in seguito. Al termine della prima fase di questo metodo, ha luogo un semplice test di ammissibilità per il problema.

Viceversa, un problema è *illimitato* (e perciò, non esiste un valore ottimo *finito*) se la regione ammissibile è *illimitata* verso più o meno infinito (a seconda, rispettivamente, che si tratti di un problema di *massimizzazione* o *minimizzazione*). In particolare, un problema è *illimitato* se gli elementi di una colonna $\boldsymbol{A}_j$ candidata ad entrare in base, sono **minori o uguali** a 0. Questo, a sua volta, significa che un incremento di quella variabile porterebbe ad un *allontanamento* dal vincolo o a nessuno spostamento (nel caso di coefficiente pari a 0, che impedirebbe tra l'altro la divisione) e che, quindi, la variabile può essere incrementata indefinitamente.

§2.4 – Esempio di applicazione

Per illustrare la tecnica appena descritta, si risolverà un breve esempio. Esaminiamo il seguente problema:

$$\text{Max. } Z = 3x_1 + 5x_2$$
$$\text{s.t.}$$
$$x_1 \leq 4$$
$$2x_2 \leq 12$$
$$3x_1 + 2x_2 \leq 18$$

Per raggiungere la **forma aumentata** e, quindi, dare l'avvio all'**algoritmo del simplesso**, è necessario inserire una **variabile slack** non-negativa per ogni vincolo. Come già detto, queste rappresentano lo *scarto* tra il valore delle variabili nel vincolo e il termine noto; sono necessarie per rendere le disequazioni in equazioni ed ottenere una **BFS** iniziale. Questa è la **forma aumentata** del problema:

$$\text{Max. } Z = 3x_1 + 5x_2$$
$$\text{s.t.}$$
$$x_1 + s_1 = 4$$
$$2x_2 + s_2 = 12$$
$$3x_1 + 2x_2 + s_3 = 18$$

A questo punto, è possibile inserire il problema nel *tableau*.

		x_1	x_2	s_1	s_2	s_3	
0)	Z	-3	-5	0	0	0	0
1)	s_1	1	0	1	0	0	4
2)	s_2	0	2	0	1	0	12
3)	s_2	3	2	0	0	1	18

Osservando il *tableau* sopra riportato, si può notare che le **variabili slack** sono **in base** e le altre, sono pari a 0 nella **BFS**.

Si noti come le colonne **slack** (ad eccezione del valore nella prima riga) formino una **base canonica** per lo spazio di ordine pari al numero dei vincoli; questo è un modo immediato per identificare le **variabili di base** e la correttezza (canonicità) del *tableau* attuale.

A questo punto, è necessario scegliere una variabile *entrante*: una variabile che, aumentata, permetta un *miglioramento* della funzione obiettivo (in questo caso, una maggiorazione, essendo un problema di massimo). Tra le variabili candidate (quelle con coefficiente *negativo* nella prima riga del *tableau*) scegliamo quella con il coefficiente *minore*, x_2.

Scelta la variabile *entrante*, si procede al **test del minimo rapporto**. Nel vincolo 1), la variabile ha coefficiente 0; non si effettua test. Il vincolo 2) e il vincolo 3) danno risultati, rispettivamente, 6 e 9; quindi, si sceglie s_2 come variabile *uscente*.

		x_1	x_2	s_1	s_2	s_3	
0)	Z	-3	-5	0	0	0	0
1)	s_1	1	0	1	0	0	4
2)	s_2	0	2	0	1	0	12
3)	s_2	3	2	0	0	1	18

E' ora necessario modificare il *tableau* per portarlo ad una forma che tenga conto delle nuove **variabili di base**. In particolare, per prima cosa si divide la *riga pivot* per l'*elemento pivot*, in questo caso la riga 2) per lo scalare 2. Dopodiché, si rende la *colonna pivot* uguale alla colonna della precedente **variabile di base** s_2, mediante semplici operazioni di riga; qui, si somma 5 volte la riga 2) alla riga 0) e si sottrae 2 volte la riga 2) alla riga 3).

Questo è il *tableau* ottenuto dopo le **operazioni di pivot**:

		x_1	x_2	s_1	s_2	s_3	
0)	Z	-3	0	0	2.5	0	30
1)	s_1	1	0	1	0	0	4
2)	x_2	0	1	0	0.5	0	6
3)	s_2	3	0	0	-1	1	6

Il **test di ottimalità** fallisce, perché è ancora possibile aumentare la variabile x_1 per ottenere un miglioramento dell'ottimo. Siccome è l'unica variabile ad offrire questa possibilità, viene scelta come nuova variabile *entrante*.

Il **test del minimo rapporto** indica che s_2, con un risultato di 2, è candidata ad uscire di base.

		x_1	x_2	s_1	s_2	s_3	
0)	Z	-3	0	0	2.5	0	30
1)	s_1	1	0	1	0	0	4
2)	x_2	0	1	0	0.5	0	6
3)	s_2	3	0	0	-1	1	6

Per normalizzare il tableau, stavolta, è necessario, nell'ordine: dividere la riga 3) per l'*elemento pivot* 3, sottrarre una volta la riga 3) alla riga 1, sommare 3 volte la riga 3) alla riga 0). Dopo aver effettuato le nuove **operazioni di pivot** sull'elemento 3, il *tableau* ha questo aspetto:

		x_1	x_2	s_1	s_2	s_3	
0)	Z	0	0	0	1.5	1	36
1)	s_1	0	0	1	0.33	-0.33	2
2)	x_2	0	1	0	0.5	0	6
3)	x_1	1	0	0	-0.33	0.33	2

Il problema supera il **test di ottimalità**; nessuna variabile può essere aumentata per ottenere un *miglioramento* della funzione obiettivo. Il *tableau* è perciò **ottimo**; il valore 36, il primo della colonna dei termini noti, è l'**ottimo** per il problema esaminato.

§2.5 – *Problemi in forma non standard*

Mediante opportuni accorgimenti algebrici, qualsiasi problema di Programmazione Lineare può essere portato alla **forma standard**. Delle differenti possibilità, alcune possono essere trattate con semplici manipolazioni; altre, necessitano di un trattamento più approfondito. Riassumiamo di seguito i casi che possono essere semplicemente ricondotti ad un equivalente problema standard.

- **Problema di minimizzazione.** In questo caso, è sufficiente ricordare che

$$\text{Min.} = \sum_{i=1}^{n} c_i x_i$$

è equivalente a

$$\text{Max.} - Z = \sum_{i=1}^{n} -c_i x_i$$

Perciò è sufficiente, tenendo conto di questa manipolazione, riportare i coefficienti nel *tableau* **con lo stesso segno** con cui compaiono nella funzione obiettivo, invece che con segno inverso. Il procedimento che segue è il medesimo. Si ha cura di notare che in questo modo, l'ottimo che si ottiene è $-Z$ e non Z; perciò, deve essere invertito per ottenere l'ottimo reale.

- **Variabili senza vincoli di non-negatività.** Come detto sopra, nel *tableau* non è possibile avere una variabile negativa (ciò porterebbe ad una **soluzione di base non ammissibile**). Perciò, è necessario manipolare le variabili che possono assumere valori negativi, in modo da ottenere un problema equivalente con variabili vincolate alla non-negatività.

Si distinguono due casi. Nel primo, un variabile è *limitata* inferiormente da un numero negativo:

$$x_i \geq L$$

con $L < 0$ (costante *numerica*). Si introduce una nuova variabile $x_i^{'}$ tale che $x_i^{'} = x_i - L$ e si operano le necessarie sostituzioni nei vincoli e nella funzione obiettivo. In questo modo, la variabile $x_i^{'}$ sarà soggetta al solo vincolo di non-negatività.

Nel secondo caso, si ha una variabile x_i *illimitata* inferiormente. Si introducono *due* nuove variabili $x_i^{'}$ e $x_i^{''}$, entrambe non-negative, tali che:

$$x_i = x_i^{'} - x_i^{''}$$

Dopodiché, si opera una sostituzione in tutti i vincoli e nella funzione obiettivo e si procede nel modo consueto. Questo metodo ha lo svantaggio di introdurre un'ulteriore variabile.

Alcune classi di problemi non possono essere immediatamente ricondotte ad un problema in **forma standard** e devono essere risolte con un procedimento particolare,

detto **metodo a due fasi**. Qui di seguito sono introdotte queste forme.

- **Vincoli di uguaglianza.** La difficoltà presentata da un vincolo di uguaglianza, consiste nel fatto che il vincolo è *già* un'equazione e, di conseguenza, non necessita di una **variabile slack** per ottenere la **forma aumentata**. Questo risulta nell'impossibilità di ottenere immediatamente una **BFS** iniziale e, quindi, di avviare il **metodo del simplesso**. Per ovviare al problema, si inserisce nel vincolo una variabile non-negativa, detta **variabile artificiale**, con coefficiente 1 e nella funzione obiettivo con coefficiente -1 (o 1, se si tratta di un problema di **minimizzazione**). Ad esempio, il problema (già in **forma aumentata**):

$$\text{Max. } Z = 3x_1 + 5x_2$$
$$\text{s.t.}$$
$$x_1 + s_1 = 4$$
$$2x_2 + s_2 = 12$$
$$3x_1 + 2x_2 = 18$$

  diventa:

$$\text{Max. } Z = 3x_1 + 5x_2 - a_1$$
$$\text{s.t.}$$
$$x_1 + s_1 = 4$$
$$2x_2 + s_2 = 12$$
$$3x_1 + 2x_2 + a_1 = 18$$

  dove a_1 è la **variabile artificiale**, maggiore di 0. Questa sarà la **variabile di base** per la **BFS** iniziale del problema. Un problema a cui siano state aggiunte una o più **variabili artificiali**, si dice in **forma artificiale**.

- **Vincoli di maggioranza**. Il primo passo per portare un problema con vincoli di maggioranza in **forma standard**, consiste nell'introdurre nel vincolo una variabile non negativa, detta **variabile surplus**, con coefficiente -1. Questa variabile rappresenta *l'eccesso* (*surplus*, per l'appunto) rispetto alla frontiera del vincolo. Il vincolo:

$$0.6x_1 + 0.4x_2 \geq 6$$

  diventa:

$$0.6x_1 + 0.4x_2 - s_3 = 6$$

  Ma la **variabile surplus** così inserita, non permette di ottenere una **BFS**, perché

non ha coefficiente pari a 1. Perciò viene aggiunta, anche qui, una **variabile artificiale**:

$$0.6x_1 + 0.4x_2 - s_3 + a_1 = 6$$

che permette di ottenere una **BFS** iniziale. Come sopra, anche questa **variabile artificiale** sarà aggiunta alla funzione obiettivo con verso opposto all'ottimo (coefficiente -1 se si tratta di un problema di massimizzazione, 1 se il problema è di minimizzazione).

- **Termini noti negativi.** Per ovviare ad un termine noto negativo, che porterebbe la **BFS** in condizioni di non ammissibilità, si moltiplica tutto il vincolo per -1. Questo ha l'effetto di modificare, oltre ai segni di tutti i termini, anche il verso del vincolo (se si tratta di una disequazione). Ad esempio, un vincolo:

$$x_1 + 2x_2 - x_3 \leq -6$$

diventa:

$$-x_1 - 2x_2 + x_3 \geq 6$$

Il vincolo viene poi trattato nel modo appropriato per la tipologia risultante (i.e. se il risultante è un vincolo di maggioranza, sono introdotte una **variabile surplus** e una **variabile artificiale**), come sopra descritto.

§2.6 – *Metodo delle due fasi*

Le operazioni su problemi non riconducibili a **forma standard** sopra descritte, portano alla cosiddetta **forma artificiale** del problema, in cui una o più **variabili artificiali** compaiono nella funzione obiettivo. Il problema d'esempio:

$$\begin{aligned}
&\text{Min. } Z = 0.4x_1 + 0.5x_2 \\
&\text{s.t.} \\
&0.3x_1 + 0.1x_2 \leq 2.7 \\
&0.5x_1 + 0.5x_2 = 6 \\
&0.6x_1 + 0.4x_2 \geq 6 \\
&x_{1\ldots2} \geq 0
\end{aligned}$$

diventa il problema artificiale:

$$\text{Max. } -Z = -0.4x_1 - 0.5x_2 + a_1 + a_2$$
$$\text{s.t.}$$
$$0.3x_1 + 0.1x_2 + s_1 = 2.7$$
$$0.5x_1 + 0.5x_2 + a_1 = 6$$
$$0.6x_1 + 0.4x_2 - s_2 + a_2 = 6$$
$$x_{1\ldots2} \geq 0, s_{1\ldots2} \geq 0, a_{1\ldots2} \geq 0$$

Si deve tenere conto che, come precedentemente accennato, le **variabili artificiali** *penalizzano* la funzione obiettivo, opponendosi al raggiungimento dell'ottimo; la loro definizione impone che esse assumano il valore 0.

Basandosi su quest'idea, la *fase uno* del **metodo delle due fasi** consiste nel *minimizzare* le **variabili artificiali**, ignorando le variabili decisionali presenti nella funzione obiettivo, che per questo problema diventa

$$\text{Min. } Z = a_1 + a_2$$

Il *tableau* iniziale è questo:

		x_1	x_2	s_1	s_2	a_1	a_2	
0)	$-Z$	0	0	0	0	1	1	0
1)	s_1	0.3	0.1	1	0	0	0	2.7
2)	a_1	0.5	0.5	0	0	1	0	6
3)	a_2	0.6	0.4	0	-1	0	1	6

Come si può notare, il *tableau* non è in **forma canonica:** due delle **variabili di base**, a_1 e a_2, compaiono nella prima riga con coefficiente pari a 1. Perciò, è necessario *normalizzare* il *tableau*, sottraendo una volta la riga 2) alla riga 0) e la riga 3) alla riga 0). Il *tableau* risultante è questo:

		x_1	x_2	s_1	s_2	a_1	a_2	
0)	$-Z$	-1.1	-0.9	0	1	0	0	-12
1)	s_1	0.3	0.1	1	0	0	0	2.7
2)	a_1	0.5	0.5	0	0	1	0	6
3)	a_2	0.6	0.4	0	-1	0	1	6

Si procede poi normalmente, con l'usuale algoritmo del simplesso, fino ad ottenere questo *tableau* ottimo per la *fase uno*:

		x_1	x_2	s_1	s_2	a_1	a_2	
0)	$-Z$	0	0	0	0	1	1	0
1)	x_1	1	0	5	0	-1	0	7.5
2)	s_2	0	0	1	1	0.6	-1	0.3
3)	x_2	0	1	-5	0	3	0	4.5

in cui tutte le **variabili artificiali** sono *fuori base* e quindi, pari a 0. E' importante notare come il *tableau* ottimo della *fase uno* costituisca un **test di ammissibilità** per il problema: se l'ottimo fosse stato *diverso* da 0, o, in altre parole, se almeno una delle **variabili artificiali** fosse rimasta *in base*, ciò significherebbe che il problema non possiede soluzioni ammissibili nelle sue sole variabili originarie.

Superato il test di ammissibilità, si procede nel modo seguente per introdurre la *fase due* dell'algoritmo. Per prima cosa, si eliminano dal *tableau* le colonne corrispondenti alle variabili artificiali e si inseriscono i coefficienti della **funzione obiettivo** originaria nella riga 0:

		x_1	x_2	s_1	s_2	
0)	$-Z$	0.4	0.5	0	0	0
1)	x_1	1	0	5	0	7.5
2)	s_2	0	0	1	1	0.3
3)	x_2	0	1	-5	0	4.5

In secondo luogo, si provvede a normalizzare il *tableau* rispetto alle variabili di base:

		x_1	x_2	s_1	s_2	
0)	$-Z$	0	0	0.5	0	-5.25
1)	x_1	1	0	5	0	7.5
2)	s_2	0	0	1	1	0.3
3)	x_2	0	1	-5	0	4.5

Questo *tableau* è già ottimo, ma se non lo fosse, andrebbe risolto con il procedimento consueto. Il valore dell'ottimo è 5.25 (si ricorda che in un problema di *minimizzazione*, l'ottimo cercato è il valore di $-Z$).

§2.7 – Conclusioni

Il **metodo del simplesso** è un metodo pratico ed efficiente per risolvere problemi di **Programmazione Lineare** in tempi brevi. Per la sua natura iterativa, può essere diviso in *fasi* ben distinte e ripetute, una o più a seconda dei dati numerici inseriti. In letteratura possono essere trovate numerose implementazioni delle differenti versioni del metodo, più o meno ottimizzate per varie specializzazioni: matrici sparse, problemi su grafi, problemi di grandissime dimensioni e così via.

Nella prossima sezione verranno riprese alcune implementazioni del **metodo del simplesso**, la maggior parte delle quali a scopo didattico.

Capitolo 3

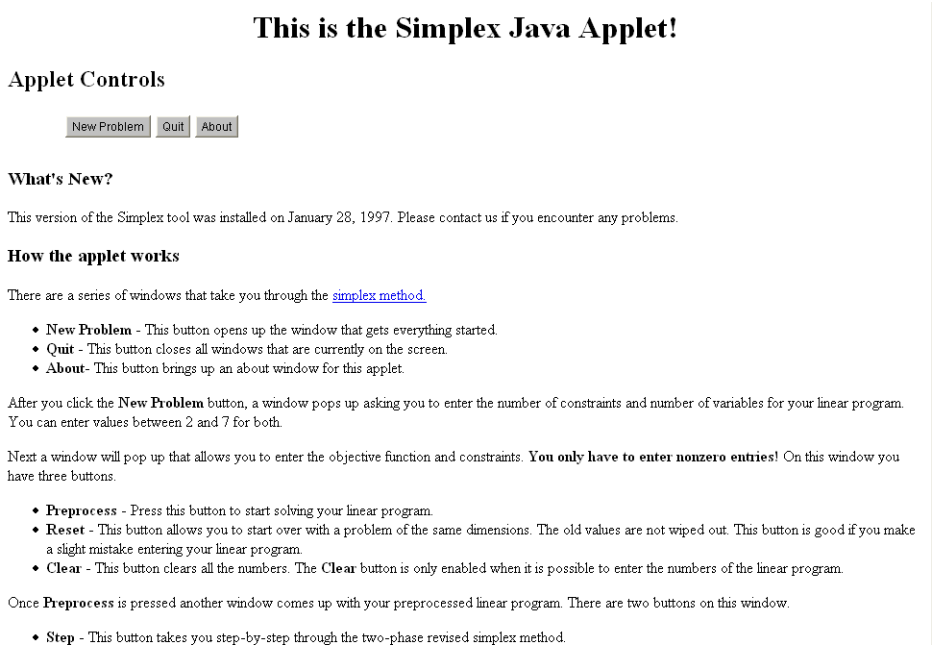
Stato dell'arte: strumenti esistenti

In questa sezione saranno presi in considerazione alcuni strumenti, la maggior parte dei quali a finalità didattiche, che implementano il **metodo del simplesso** presentato nel capitolo precedente. La gran parte di questi si presenta sotto forma di *applet* in linguaggio Java, facilmente utilizzabili tramite il *web*. Dopo un'attenta ricerca, sono stati selezionati quattro *tool* di cui saranno analizzate caratteristiche e funzionalità.

§3.1 – The Simplex Java Applet

Questo *applet*, scritto (come facilmente intuibile)in linguaggio Java, è utilizzabile dalla pagina web http://www.informatica.us.es/~calvo/pl/ejjavasimplex/Simplex.htm ed è fornito da **Mathematics and Computer Science Division – Argonne Labs.**

Il programma risolve problemi di **Programmazione Lineare continua**, limitati tra 2 e 7 variabili e tra 2 e 7 vincoli, di *massimizzazione* o *minimizzazione.* Tutte le variabili sono assunte come non-negative e i vincoli possono essere di maggioranza, minoranza o uguaglianza (è implementato il simplesso a due fasi).



This is the Simplex Java Applet!

Applet Controls

New Problem | Quit | About

What's New?

This version of the Simplex tool was installed on January 28, 1997. Please contact us if you encounter any problems.

How the applet works

There are a series of windows that take you through the simplex method.

- **New Problem** - This button opens up the window that gets everything started.
- **Quit** - This button closes all windows that are currently on the screen.
- **About**- This button brings up an about window for this applet.

After you click the **New Problem** button, a window pops up asking you to enter the number of constraints and number of variables for your linear program. You can enter values between 2 and 7 for both.

Next a window will pop up that allows you to enter the objective function and constraints. **You only have to enter nonzero entries!** On this window you have three buttons.

- **Preprocess** - Press this button to start solving your linear program.
- **Reset** - This button allows you to start over with a problem of the same dimensions. The old values are not wiped out. This button is good if you make a slight mistake entering your linear program.
- **Clear** - This button clears all the numbers. The **Clear** button is only enabled when it is possible to enter the numbers of the linear program.

Once **Preprocess** is pressed another window comes up with your preprocessed linear program. There are two buttons on this window.

- **Step** - This button takes you step-by-step through the two-phase revised simplex method.

Fig. 1: Pagina iniziale di "Simplex Java Applet"

§3.1.1 - Caratteristiche

Nella pagina ospitante l'applet, visualizzata in **Fig. 1**, compaiono i controlli per l'avvio e la terminazione dell'*applet* e una breve guida all'utilizzo.

Il pulsante "**New Problem**" fa comparire una piccola finestra, riportata in **Fig. 2**. Qui sono presenti due campi, uno per il numero di variabili e uno per il numero di vincoli.

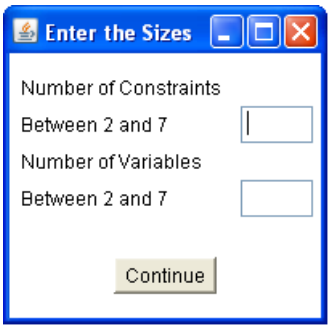


Fig. 2: Finestra di inserzione dati

Si può inserire al minimo 2 e al massimo 7 in ciascuno dei due campi, ma non vi è alcun controllo sul tipo di dati inseriti. L'inserimento di simboli non numerici non provoca nessun *feedback*: l'errore è semplicemente ignorato. L'*input* di numeri al di sopra o al di sotto del limite consentito, invece, riporta alla stessa finestra con una segnalazione sul campo errato.

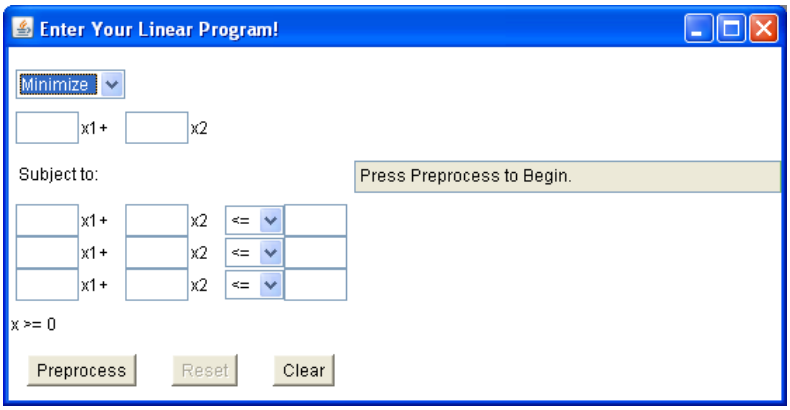


Fig. 3: Finestra di inserzione parametri e vincoli

L'inserimento di dati corretti, invece, permette l'apertura della finestra successiva (**Fig. 3**), con i campi per l'accettazione dei dati del problema vero e proprio.

In quest'ultima finestra, sono presenti *drop-down menu* che permettono la scelta tra massimizzazione e minimizzazione, nonché il tipo dei vincoli. Tale tipologia di menu restringe le possibilità dell'utente, impedendo eventuali errori nell'inserzione dei dati. I campi sono tanti quante le dimensioni inserite nella precedente finestra.

Nei campi per l'*input* dei parametri, invece, il medesimo accorgimento non è usato:all'utente viene lasciata libertà di inserire valori a piacimento. Qui, un tentativo di inserimento di simboli non numerici provoca un messaggio nella "barra di stato" a destra, che indica la prima casella in cui è stato rilevato un errore. Caselle vuote sono interpretate come uno 0.

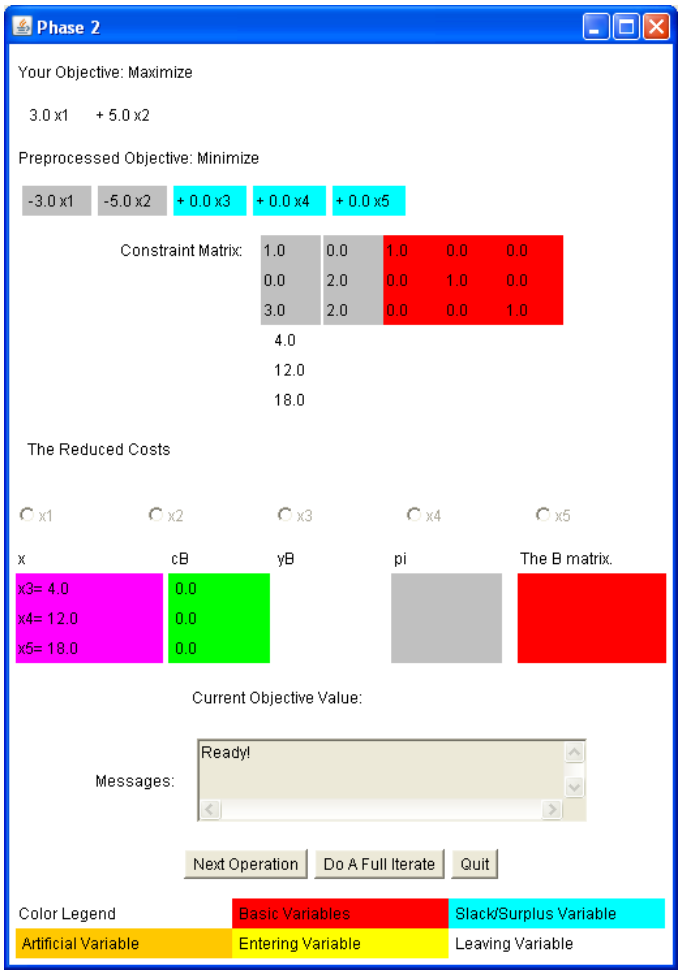


Fig. 4: Finestra di risoluzione

La pressione del tasto "**Preprocess**" porta alla finestra successiva, in cui il problema sarà risolto.

Quest'ultima, rappresentata in **Fig. 4**, risulta organizzata in tre parti principali:

- la funzione obiettivo, nella forma inserita e nella **forma standard** (che qui è un problema di minimizzazione, con vincoli di minoranza), la tabella dei vincoli (parte del *tableau*) e, al di sotto, i valori dei **termini noti**;
- una riga etichettata "*The Reduced Costs*" (coefficienti di costo ridotto, inizialmente vuota), dei *radio button* etichettati con le **variabili decisionali** e alcune tabelle, variamente colorate, con dati che saranno di volta riempiti durante l'esecuzione e il valore ottimo corrente, anch'esso inizialmente vuoto;
- un campo testuale che conterrà, di volta in volta, i messaggi di stato, al di sotto di esso i comandi (rispettivamente "**Prossima operazione**", "**Esegui**

un'iterazione completa" ed "**Esci**") e una breve legenda per alcuni dei colori utilizzati nella finestra.

§3.1.2 –Test e risultati

Il programma mostra all'utente i passi dell'algoritmo, se questi ne richiede la visualizzazione premendo successivamente il tasto "**Next operation**", oppure un'intera iterazione premendo "**Do a Full Iterate**". Entrambi **non permettono in alcun modo** l'interazione durante il procedimento. All'utente sono semplicemente presentati i **risultati** della computazione.

Solo nella fase di scelta della variabile *entrante*, come mostrato in **Fig. 5**, è prevista una parziale interazione. Se ci sono più candidate alla scelta, si può selezionare una variabile, ma solo tra quelle con costo ridotto negativo.

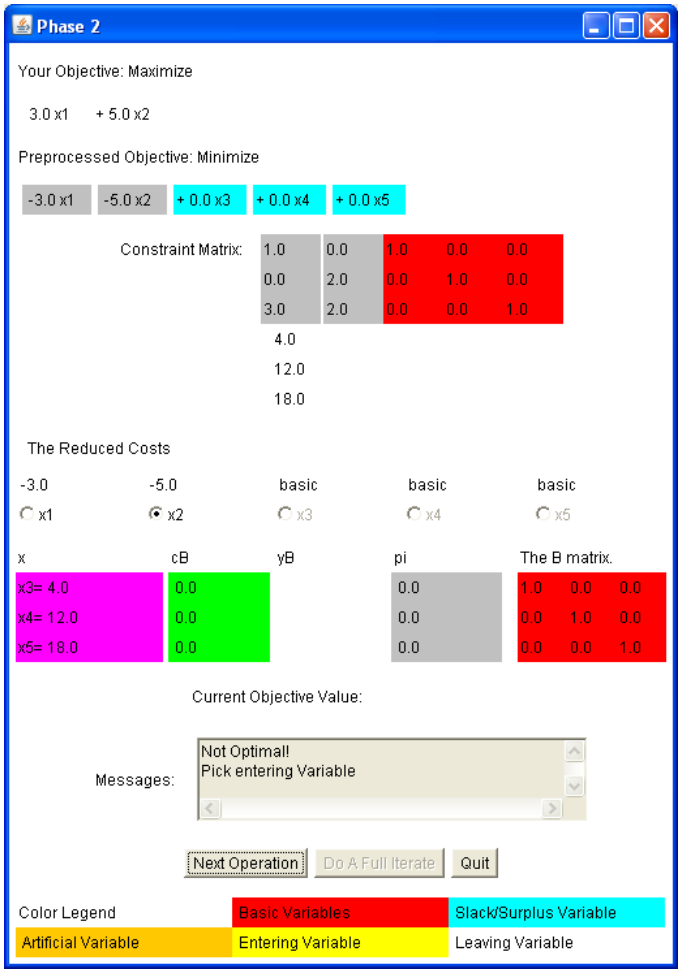


Fig. 5: Finestra di risoluzione, che presenta un'interattività molto limitata

L'obiettivo è presentato (e aggiornato) separatamente dal *tableau*, mentre la colonna dei termini noti (e, di conseguenza, il valore e l'identità delle variabili di base) è situata

in basso a sinistra, nella tabella color fucsia. Quest'ultima permette di identificare la **BFS** corrente con relativa facilità. E' interessante notare come i termini noti del problema iniziale, sono mantenuti inalterati durante l'esecuzione del problema.

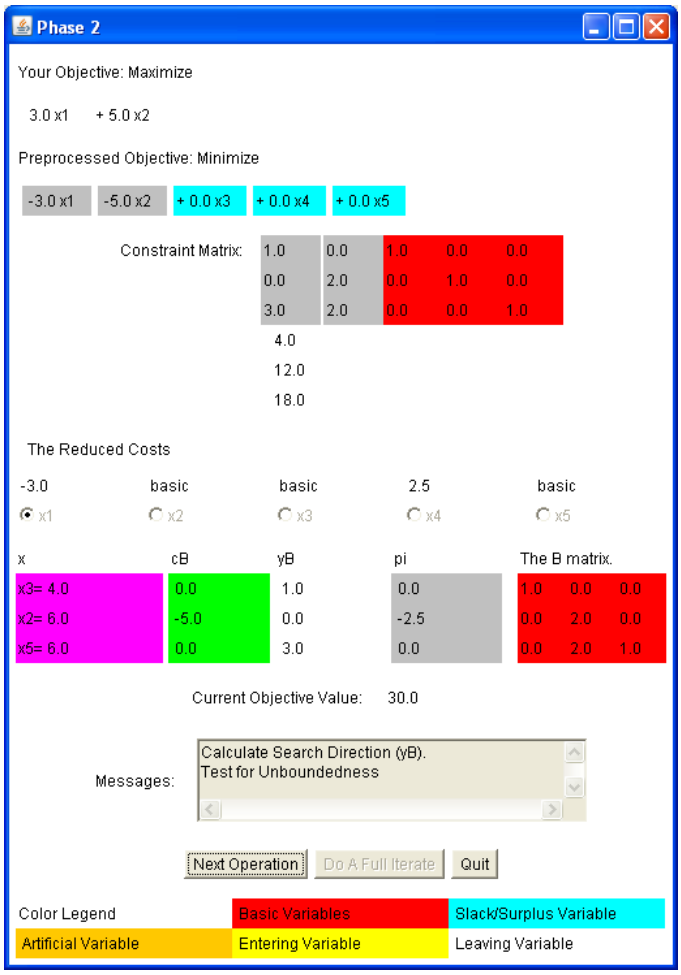


Fig. 6: Esecuzione silenziosa dei test

I test presenti in ogni problema (ottimalità e *boundness*) sono anch'essi effettuati in maniera "silenziosa" o quasi: ne è dato solo l'esito, al momento di effettuarli.

I **problemi artificiali** sono trattati in modo analogo. Vengono presentate le **variabili artificiali**, in colore diverso dalle altre, in coda alla funzione obiettivo "preprocessata" (secondo il gergo adottato dall'*applet*); alla fine della **fase uno**, le variabili artificiali sono eliminate e viene effettuato il test di ammissibilità in maniera "silenziosa" (senza *feedback*), analogamente a prima. Anche i vari casi di terminazione dell'algoritmo (problema illimitato, impossibile od ottimo) sono presentati senza aggiungere spiegazioni o cause.

In definitiva, sebbene l'interfaccia non risulti di immediata lettura, tutte le informazioni

sono presenti. I punti di forza dell'*applet* sono l'**utilizzo della colorazione** per distinguere le varie componenti del *tableau*, sebbene spezzato lungo l'interfaccia, e la **possibilità di visualizzare i singoli passi** dell'algoritmo, oltre all'iterazione completa. Sebbene manchi quasi del tutto la possibilità di interagire durante le varie fasi della risoluzione, questo strumento presenta le **maggiori affinità** con le caratteristiche di semplicità d'uso e completezza nella presentazione, che ci si è prefissi di raggiungere con il programma sviluppato nella presente tesi.

§3.2 – *Simplex Applet*

Questo *applet* Java è scaricabile alla pagina http://vinci.inesc.pt/lp/ ed è fornito da **Algos (***Algorithms for Optimization and Simulation***)**, Lisbona, un gruppo di ricerche dedito allo sviluppo di algoritmi di ottimizzazione e simulazione.

Il *tool* risolve problemi di **Programmazione Lineare continua**, che necessitino di algoritmo ad una e a due fasi, effettua la conversione da **primale** a **duale** ed implementa anche il **simplesso duale**.

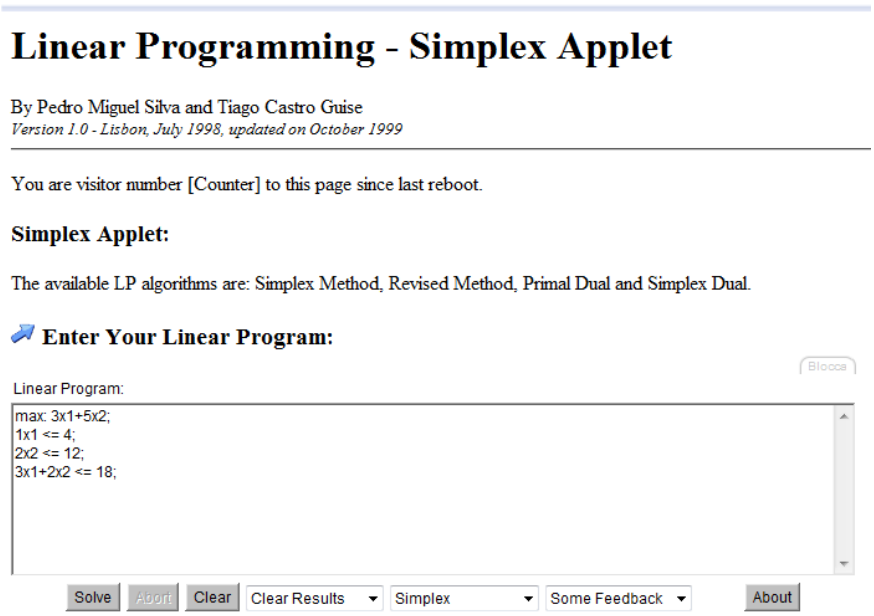
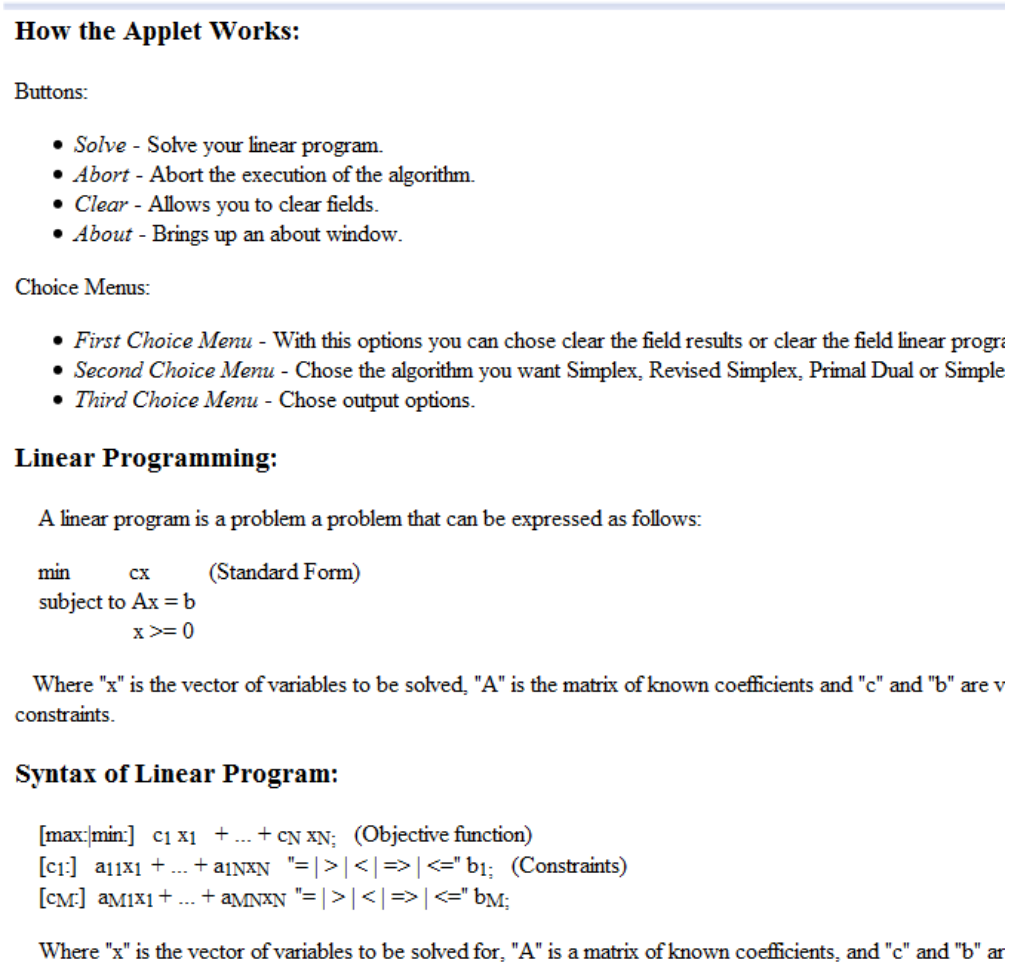


Fig. 7: Simplex Applet - campo di inserzione dati

§3.2.1 - Caratteristiche

L'*applet* è contenuto in una sola pagina *web*, a differenza del precedente. All'utente è presentato un campo testuale inizialmente vuoto, senza "maschere" predefinite, più

alcuni controlli sotto di esso. Scorrendo la pagina, si trovano un ulteriore campo testuale vuoto e, sotto di esso, una serie di istruzioni non chiarissime sulla sintassi da utilizzare.



How the Applet Works:

Buttons:

- *Solve* - Solve your linear program.
- *Abort* - Abort the execution of the algorithm.
- *Clear* - Allows you to clear fields.
- *About* - Brings up an about window.

Choice Menus:

- *First Choice Menu* - With this options you can chose clear the field results or clear the field linear progra
- *Second Choice Menu* - Chose the algorithm you want Simplex, Revised Simplex, Primal Dual or Simple
- *Third Choice Menu* - Chose output options.

Linear Programming:

A linear program is a problem a problem that can be expressed as follows:

min cx (Standard Form)
subject to Ax = b
x >= 0

Where "x" is the vector of variables to be solved, "A" is the matrix of known coefficients and "c" and "b" are v constraints.

Syntax of Linear Program:

[max:|min:] $c_1 x_1 + ... + c_N x_N$; (Objective function)
[c_1:] $a_{11}x_1 + ... + a_{1N}x_N$ "= | > | < | => | <=" b_1; (Constraints)
[c_M:] $a_{M1}x_1 + ... + a_{MN}x_N$ "= | > | < | => | <=" b_M;

Where "x" is the vector of variables to be solved for, "A" is a matrix of known coefficients, and "c" and "b" ar

Fig. 8: Istruzioni per l'uso di Simplex Applet.

§3.2.2 – Test e risultati

Al di sotto delle istruzioni, un problema d'esempio permette di chiarire ulteriormente quale sia la sintassi accettata dal *parser* del campo testuale. Una volta inserito un problema, è possibile decidere l'azione da effettuare alla pressione del tasto "**Solve**": se risolvere il problema con il metodo del **simplesso**, con il **simplesso rivisitato** (che fa largo uso della moltiplicazione di matrici, mentre l'originale metodo sfrutta le operazioni elementari di riga), se trovare il **duale** del problema o se risolvere il **simplesso duale**.

Fig. 9: Risultati dell'esecuzione di Simplex Applet

La **Fig. 9** presenta il risultato dell'esecuzione sul problema inserito in **Fig. 7**.La pressione del tasto **Solve**, con il **metodo del simplesso** selezionato come azione predefinita, fa comparire alcuni messaggi relativi al problema inserito, alle iterazioni effettuate e il risultato, con i valori finali delle variabili decisionali e dell'ottimo; nient'altro. Non vengono visualizzati né i passaggi intermedi, né il percorso effettuato tra di essi.

Alcune opzioni, in particolare la ricerca del **duale** e la risoluzione utilizzando il **simplesso duale**, sono specifiche di questo *applet*, ma non rispondono allo scopo di un programma didattico. Anche i risultati di queste opzioni, non presenti in altri programmi analoghi, sono presentati senza ulteriori spiegazioni. La **totale mancanza di interazione**, anche sotto la forma parziale di presentazione dei risultati intermedi, non ne fa un buon candidato su cui basare uno strumento didattico; si tratta, più che altro, di un **risolutore**.

§3.3 – LP Explorer

Il programma è disponibile alla pagina http://www.maths.ed.ac.uk/LP-Explorer/ ed è stato sviluppato da un team ridotto presso l'**Università di Edimburgo**.

Risolve problemi di **Programmazione Lineare continua** di qualsiasi dimensione (o più verosimilmente, di dimensioni limitate dalla memoria allocabile dal browser e dalla dimensione dei tipi di dati usati nell'implementazione) con soli **vincoli di minoranza** e **termini noti maggiori o uguali a 0**.

Caratteristica più importante, è l'implementazione del grafico per problemi di due variabili; inoltre, una volta risolto il problema inserito, il *tool* permette di effettuare l'**analisi di sensitività** (con immediata visualizzazione, se il grafico è disponibile) sui termini noti e sui coefficienti dei vincoli.

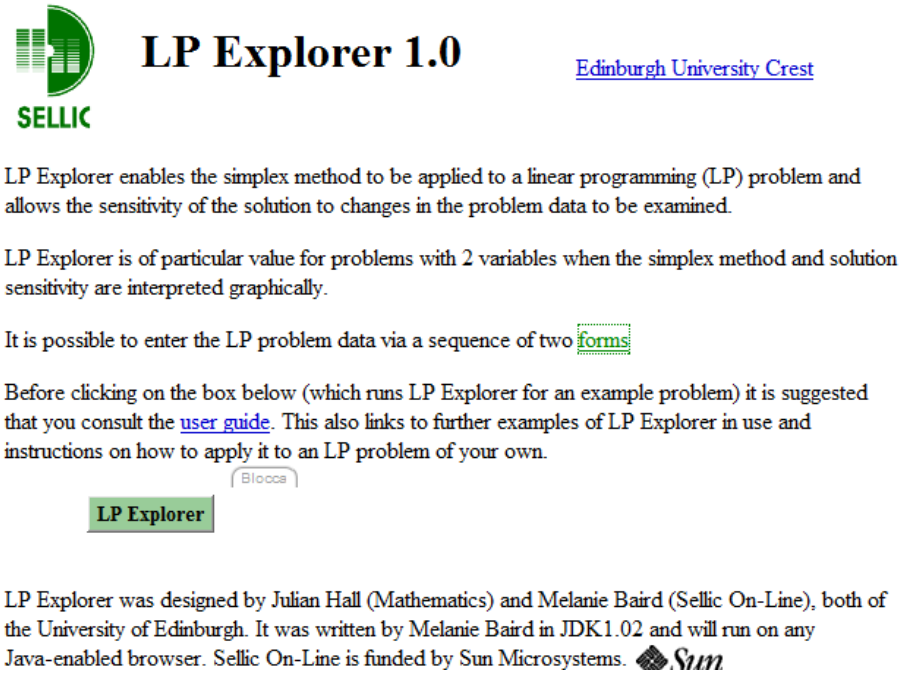


Fig. 10: LP Explorer - prima pagina

§3.3.1 - Caratteristiche

Inizialmente (**Fig. 10**), viene proposta all'utente un'implementazione dell'*applet* e la possibilità di visualizzare direttamente la finestra di risoluzione, oltre alla pagina per l'inserzione di un problema personalizzato, proposta nella **Fig. 11.**

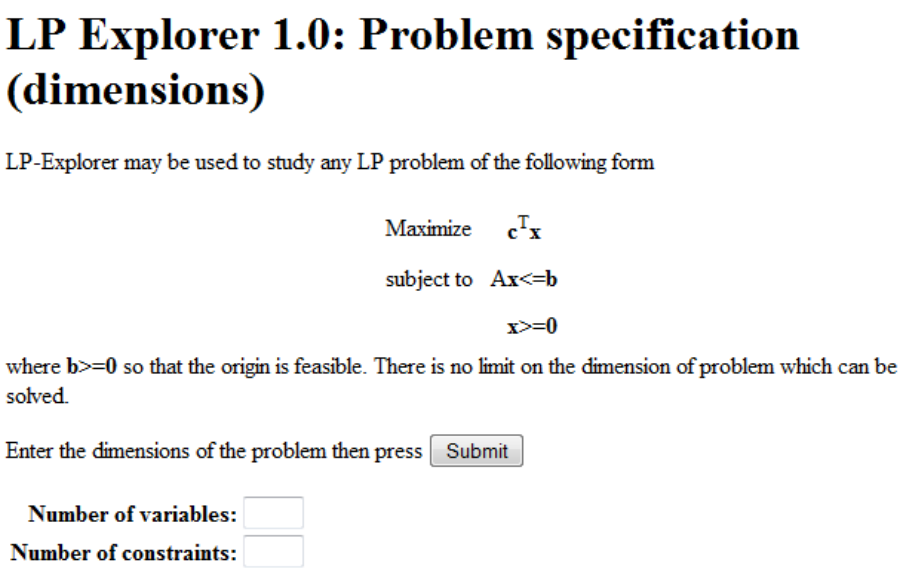


Fig. 11: Pagina di inserimento delle dimensioni di LP Explorer

A questa si accede seguendo il link "**Forms**". E' possibile prendere visione dei tipi di problemi risolti dal programma ed inserire il numero di variabili e di vincoli. Qui, la nota

informa l'utente che non è possibile inserire problemi con vincoli che non siano di minoranza, né con termini noti negativi. Il controllo sui campi numerici è effettuato senza *feedback* visibile; se vengono inseriti simboli non numerici, questi sono scartati e viene assunto il valore 1. Ciò può essere fonte di problemi, soprattutto nel caso di errori di digitazione.

La pressione del tasto "**Submit**" porta alla pagina successiva, rappresentata in **Fig. 12**, in cui è richiesto di inserire i dati del problema desiderato. Sono naturalmente presenti solo i campi corrispondenti alle quantità introdotte nella pagina precedente.

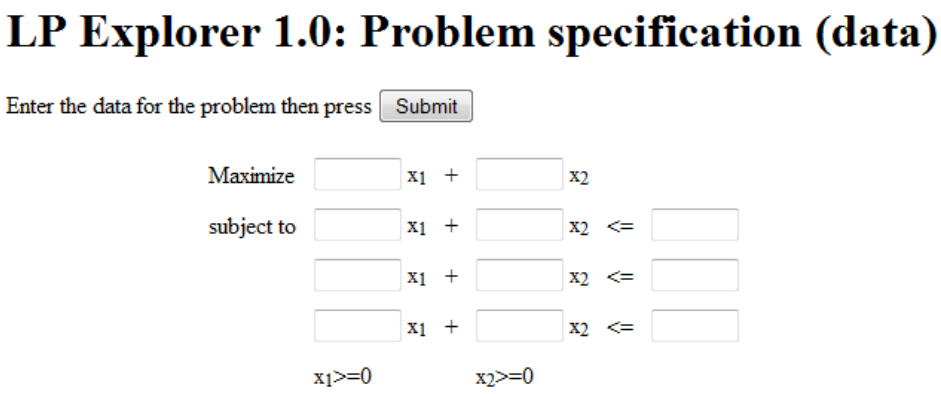


Fig. 12: Pagina di inserimento dei dati di LP Explorer

Qui, in caso di errori di digitazione, viene assunto uno 0 nella posizione non interpretabile. Per i **termini noti**, (stavolta, anche nel caso di numeri negativi), si applica la stessa regola.

La finestra del risolutore, visualizzata in **Fig. 13**, è una delle più complete e fruibili tra vari strumenti esaminati. Presenta, in modo ordinato e facilmente visibile, il problema inserito in un pannello nella parte alta, due *tableau* nel pannello centrale, i comandi della finestra sotto di essi e, per i problemi a due variabili, il **grafico del problema** a sinistra.

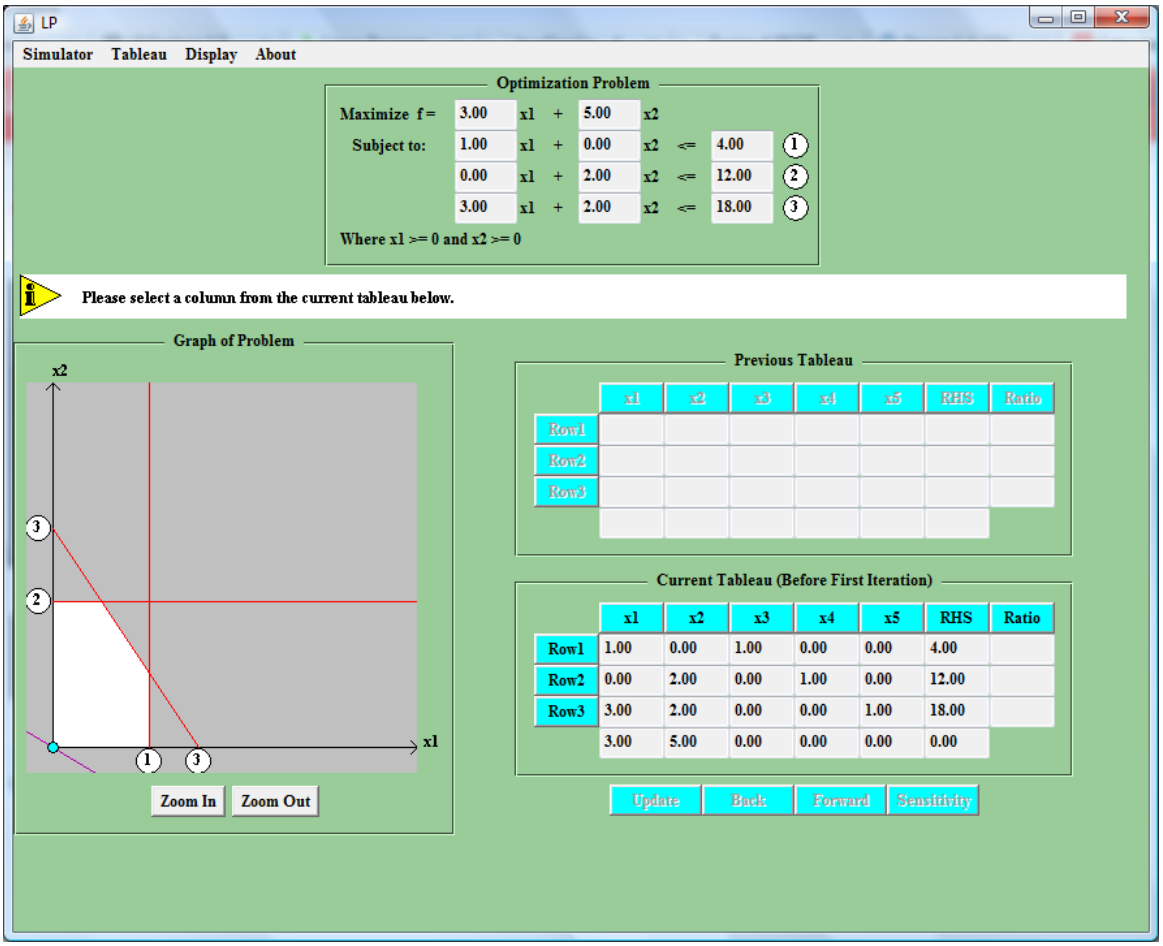


Fig. 13: Risolutore di LP Explorer

Ogni pannello è corredato con utili informazioni, sfruttate nel corso della risoluzione del problema. I vincoli sono affiancati da un numero identificativo cerchiato di bianco e facilmente visibile, ripetuto in corrispondenza delle rette nel grafico (se presente) e che permette di associare una retta ad un vincolo; i *tableau* (di cui quello superiore, inizialmente vuoto, servirà a contenere l'iterazione precedente) sono etichettati con i nomi delle variabili e riportano, oltre alle usuali informazioni, anche i rapporti tra gli elementi nella colonna dei termini noti e nella colonna della **variabile uscente** attuale. Il secondo *tableau* riporta l'iterazione corrente. Questo programma usa, come **forma standard**, un problema di *minimizzazione* con vincoli di minoranza (il che ha l'effetto di invertire di segno i coefficienti nella riga obiettivo, rispetto a una **forma standard** di *massimizzazione*) e pone la colonna dei termini noti a destra del *tableau* e la riga obiettivo al di sotto.

§3.3.2 – Test e risultati

Lo strumento presenta un certo grado di **interattività**, seppur limitata alla scelta della variabile entrante od uscente. Una barra di stato, situata subito al di sotto del pannello

con il problema, risponde alle possibili scelte dell'utente indicando di volta in volta se si tratta di una scelta esatta, di una scelta sbagliata e il perché, sia durante la scelta della **variabile entrante** che della **variabile uscente**. Se presente, sul grafico vengono **evidenziati i vertici** presi in esame con l'attuale scelta di variabili, in colori diversi a seconda che si tratti di una scelta sensata o meno, e la retta rappresentante la funzione obiettivo viene rilocata a seconda dei nuovi valori assunti; particolare, questo, molto gradito in quanto mostra un riscontro immediato degli effetti e del significato geometrico della **progressione tra vertici** propria del **metodo del simplesso**.

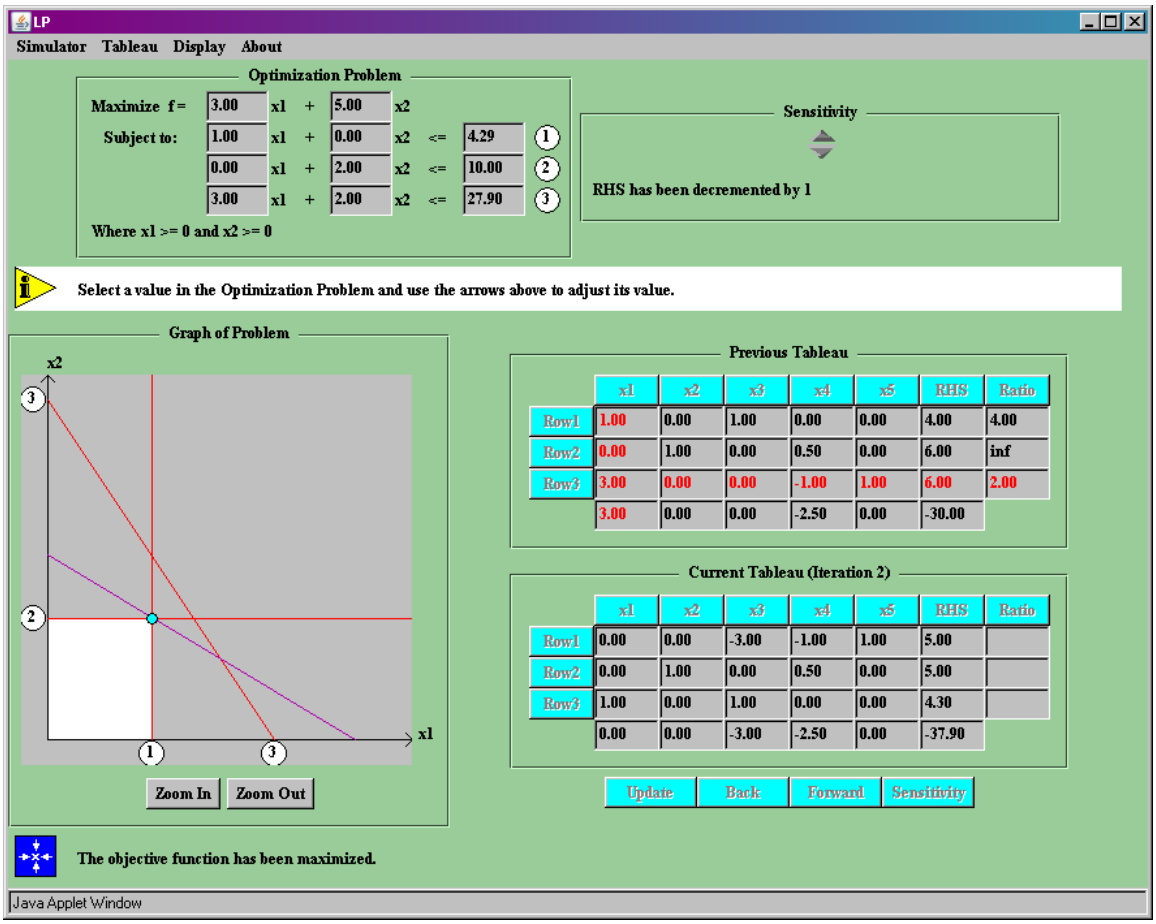


Fig. 14: Analisi di sensitività in LP Explorer

Al raggiungimento dell'ottimo (peraltro non indicato chiaramente, anche se visibile nel *tableau*), è possibile accedere ai controlli dell'**analisi di sensitività**. E' possibile modificare (mediante appositi pulsanti, non immettendo direttamente valori) i **termini noti** dei vincoli**;** i cambiamenti sono immediatamente riflessi nel *tableau* e, se presente, nel grafico. Questa fase è visibile nella **Fig. 14**.

Si tratta di un programma ben studiato e, limitatamente alla sua offerta, completo.

Mostra tutto il suo potenziale nei problemi di **Programmazione Lineare** a due variabili, per i quali è possibile visualizzare il **grafico** (completo di **regione ammissibile** e modifica della funzione obiettivo passo per passo) ed è l'unico *applet* tra quelli esaminati che, una volta risolto il problema, permette di effettuare una parziale **analisi di sensitività**. D'altro canto, manca la possibilità di risolvere ed esaminare **problemi artificiali**.

La buona **visualizzazione grafica** e la possibilità di **scegliere le variabili entranti od uscenti** cliccando sulle corrispondenti colonne, sono i punti di forza di questo programma e saranno ripresi nel progetto in corso.

§3.4 - *Java Simplex*

Questo *applet*, disponibile alla pagina http://www.benve.org/JSimplex/index.htm, è stato privatamente sviluppato.

Risolve problemi di **Programmazione Lineare continua**, con il **metodo delle due fasi** se necessario; le variabili sono assunte **maggiori di zero** e il problema, nella forma standard, è di **massimizzazione**.

§3.4.1 - Caratteristiche

L'interfaccia è ridotta all'essenziale. L'*applet* è contenuto in una sola pagina, in uno spazio più grande di quello inizialmente occupato dall'interfaccia; questa presenta, oltre alle caselle del *tableau*, i controlli che permettono la risoluzione del problema in diversi modi. L'interfaccia è rappresentata in **Fig. 15**.

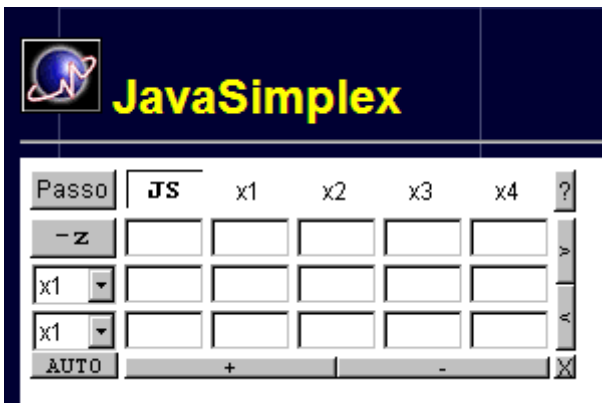


Fig. 15: Prima schermata di JavaSimplex

I tasti sul bordo destro e inferiore del *tableau*, etichettati con "**>**" e "**<**" a destra, "**+**" e "**-**" in basso0, permettono il **ridimensionamento automatico** dello spazio dedicato al problema; in particolare, rispettivamente, il numero di variabili e di vincoli. Si tratta di un'innovazione rispetto agli strumenti precedentemente esaminati, in quanto permette di eliminare il passaggio di inserimento delle dimensioni del problema.

L'interfaccia minimale risulta **poco leggibile**, ma soprattutto i metodi e i comportamenti adottati dal programma sono poco trasparenti. Un piccolo spazio nell'angolo in alto a sinistra comunica messaggi di stato, in codici di tre lettere, come esemplificato in **Fig. 16**.

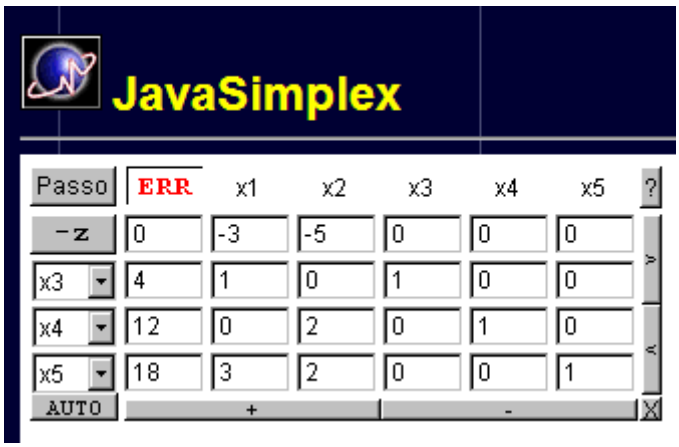


Fig. 16: Inserimento di un problema

Sotto il programma, una breve guida (**Fig. 17**) fornisce le istruzioni di funzionamento del programma: lo scopo dei vari pulsanti, il significato dei messaggi d'errore. Un problema d'esempio permette di capire in che modo inserire il *tableau*. Scorrendo la pagina, si possono esaminare altri problemi esemplificativi, di complessità crescente.

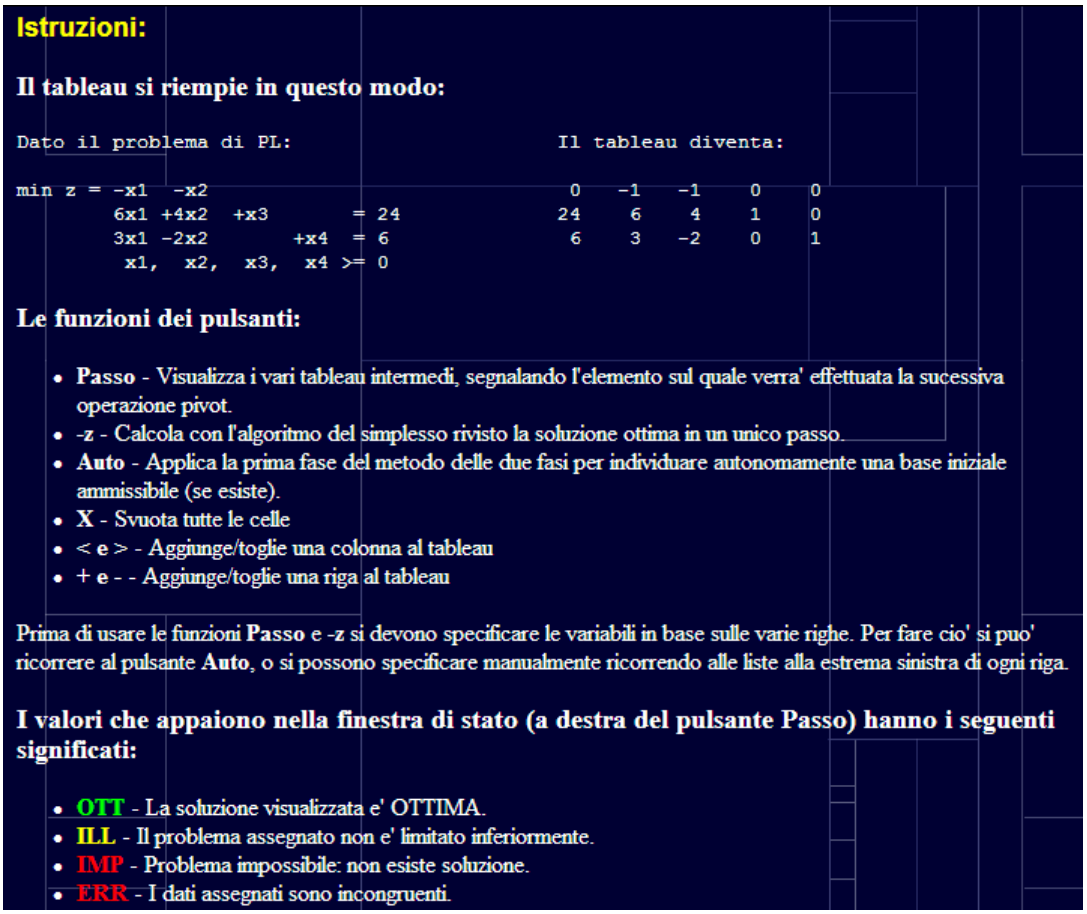


Fig. 17: Istruzioni di JavaSimplex

§3.4.2 – Test e risultati

A differenza degli altri strumenti, questo richiede che il ***tableau* iniziale venga direttamente inserito** per avviare la computazione. I dati inseriti stabiliscono la forma del problema e dei vincoli; non viene detto, ma si deduce dagli esempi che il problema in **forma standard**, come precedentemente accennato, è di *massimizzazione* e i vincoli sono di **minoranza**. La colonna dei termini noti è a sinistra del *tableau*.

E' necessario inserire *tutti* i valori iniziali, compresi gli zeri. Non viene specificato, ma l'algoritmo assume probabilmente le ultime variabili, in numero pari ai vincoli, come **variabili slack**, **surplus** o **artificiali**, a seconda del valore inserito. Ogni riga, tranne la prima, possiede alla sua sinistra un *drop-down* menu contenente le etichette di tutte le variabili inserite (pari, cioè, al numero di colonne, escludendo la prima), che è possibile utilizzare per stabilire le variabili di base. In alternativa, si può omettere questo passaggio e premere "**Auto**" per ottenere un *tableau* non normalizzato, a vari livelli di risoluzione. Se si cerca di inserire valori non numerici, l'*applet* comunica errore.

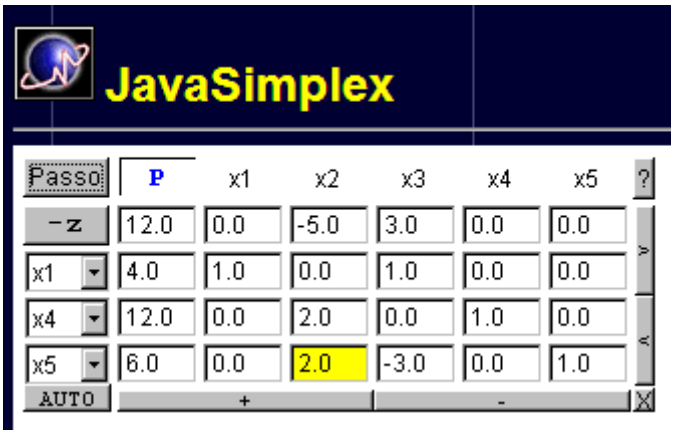


Fig. 18: Iterazione in JavaSimplex

Quando il *tableau* è completo, l'utente può premere "**Passo**" per iterare l'algoritmo del simplesso, fino alla risoluzione. I messaggi di stato comunicano "Pivot" durante le iterazioni, come rappresentato in **Fig. 18**, e uno dei tre possibili messaggi di termine alla fine ("Ottimo", "Impossibile" o "Illimitato"). I risultati sono presentati **senza interazione alcuna**, né per la scelta delle variabili entranti o uscenti, né per il calcolo dei *tableau* successivi.

La pressione del tasto "**-Z**", invece, restituisce direttamente il *tableau* ottimo, calcolato (a quanto asserisce la guida rapida) con il **metodo del simplesso rivisitato**.

Dall'analisi effettuata si può dedurre che questo programma non possa essere usato a fini didattici, ma solo illustrativi. Una sua caratteristica interessante è la possibilità di **definire le dimensioni del problema** agendo direttamente sul *tableau*. Richiede, inoltre, che l'utente sia in grado di inserire un *tableau* iniziale, operazione complessa in quanto non sono bene illustrate le modalità e le convenzioni sfruttate da questo *tool*. Nasce così il rischio di inserire, involontariamente, **un problema diverso da quello inteso,** che può portare a risultati fuorvianti. L'interazione con l'utente è veramente ridotta al minimo indispensabile.

§3.5 – *Considerazioni generali sui* tool *analizzati*

I programmi fin qui esaminati rappresentano solo un campione, estratto in base alla maggiore popolarità, dei possibili strumenti per la risoluzione del **metodo del simplesso** presenti in letteratura. Sebbene nessuno di essi sia stato creato con gli obiettivi del presente progetto, è stato tuttavia possibile estrarre da ciascuno di essi,

alcune **caratteristiche** rilevanti, che sono state riportate nel progetto finale. Analogamente, le mancanze dei vari programmi hanno suggerito nuove funzionalità da integrare nello strumento sviluppato, che verrà descritto in dettaglio nel prossimo capitolo.

Capitolo 4

Simply

In questo capitolo verrà descritto il software sviluppato nel corso del progetto. In particolare, ne saranno evidenziate le caratteristiche innovative rispetto agli strumenti presentati nel capitolo precedente, ne saranno mostrate le differenti funzionalità implementate e infine le caratteristiche principali, sia a livello architetturale che di interfaccia. Il *tool* prende il nome di ***Simply***.

§4.1 – Strumenti e ambiente di sviluppo

Il linguaggio utilizzato nello sviluppo è **Java J2SE** *(Java 2 Standard Edition)*, con *runtime* versione **1.6.0_12**. La versione più aggiornata è scaricabile alla pagina http://www.java.com/it/download/index.jsp.

L'ambiente di sviluppo integrato usato è **Eclipse Classic for Java Developers 3.4.2**. Non è stato necessario scaricare separatamente un **JDK** (*Java Development Kit*) in quanto il pacchetto multipiattaforma **Eclipse** si appoggia sulle librerie precompilate fornite dal **JRE**. Il pacchetto di installazione per la piattaforma è disponibile nella pagina http://www.eclipse.org/downloads/.

Il software è stato testato con successo sotto gli ambienti seguenti:

- **Apple Mac OS X Leopard**, versione **10.5.6**.
- **Microsoft Windows 7 Beta 1**, edizione **Ultimate 64 bit (x86_64)**.
- **Microsoft Windows Vista**, edizione **Business 32 bit, Service Pack 1**.
- **Microsoft Windows XP Professional**, **Service Pack 3**.

§4.2 – Caratteristiche principali

Simply è stato creato conservando le caratteristiche positive dei prodotti già esistenti, aggiungendo funzionalità innovative e presentando il tutto in una veste che fosse **di facile fruizione**, per renderne l'esperienza d'uso semplice e gradevole, pur mantenendo una precisa **utilità nella didattica**.

Lo scopo principale del programma è quello di **aiutare** lo studente nella risoluzione di problemi di **Programmazione Lineare**, ponendo domande e testando le soluzioni immesse. **Simply** può risolvere problemi di **Programmazione lineare continua** di dimensioni limitate, fino a **7 variabili** e altrettanti **vincoli**; questi ultimi possono essere di **maggioranza, minoranza** oppure **uguaglianza**. Il problema può essere di *massimizzazione* o *minimizzazione* e può avere termini noti negativi.

Il programma implementa il **metodo del simplesso a due fasi** con eliminazione di Gauss-Jordan. Fornisce anche brevi, essenziali, indicazioni sullo stato di esecuzione del **metodo del simplesso**. Si è cercato di non sovrapporre queste informazioni aggiuntive alle competenze che lo studente deve dimostrare in maniera autonoma.

Tramite il *tool* è possibile, per problemi a due variabili, visualizzare il **grafico del problema** completo dei vincoli, con regione ammissibile evidenziata ed alcune possibili rappresentazioni della **funzione obiettivo**, passanti per l'incontro tra i vincoli e l'asse delle ordinate, ove possibile. Quest'ultimo accorgimento ha la funzione di mostrare l'inclinazione della funzione obiettivo, senza rivelare immediatamente la posizione dell'ottimo.

Per soddisfare il requisito di **portabilità**, lo strumento viene distribuito attraverso la Rete sotto forma di archivio **Jar** eseguibile; ovvero, un archivio di classi **Java** precompilate e risorse (ad es. icone, file di testo) utilizzate dal programma stesso. In questo modo, può essere immediatamente eseguito su ogni piattaforma che disponga di Java Virtual Machine. Per funzionare, **Simply** necessita di **JRE** in versione **1.6.0_12** o superiore.

§4.3 – Innovazioni

Il *tool* **Simply** introduce una serie di *novità* rispetto agli strumenti precedentemente esaminati. Di seguito, tali innovazioni sono riportate nei particolari.

§4.3.1 – Interattività

L'interattività è il valore aggiunto più importante di **Simply**. Tutti i programmi precedentemente esaminati, ne erano sprovvisti o provvisti in quantità minime, del tutto insufficienti per un'esperienza didattica che coinvolgesse l'utente. Al contrario, con **Simply** si è voluto creare un ambiente in cui **l'interazione** non fosse solo incoraggiata, ma *necessaria* per lo svolgimento del programma.

L'idea chiave, seguita durante l'implementazione, è la seguente: lo studente di Ricerca Operativa che utilizza questo programma, **deve risolvere il problema autonomamente** e solo in seguito comunicare i suoi risultati. Lo strumento, allora, esaminerà la correttezza o meno della risposta e cercherà di dare una minima spiegazione sul perché del risultato.

Simply permette inoltre di controllare se non solo la soluzione del problema, ma anche il proprio *procedimento*, sia giusto. Questo può consentire allo studente di evitare, ad esempio, di procedere nella risoluzione di un problema mediamente complesso, solo per accorgersi che la propria soluzione non corrisponde a quella fornita dal libro di testo. **Simply**, al contrario, permette di trovare l'errore nel momento in cui viene commesso, evitando perdite di tempo dovute alla ricerca dell'errore nei passaggi precedenti.

$4.3.2 – Valutazione

Per rendere accessibile e utile il programma anche a chi si avvicina alla Programmazione Lineare, come ad esempio studenti di Ricerca Operativa nelle prime settimane del corso, è stata introdotta la possibilità di **richiedere la soluzione** di un passaggio, utile per permettere ad uno studente in difficoltà di procedere nel problema, piuttosto che ricominciare da capo o essere costretto a fermarsi. Al contrario, chiedendo la soluzione di un passaggio particolarmente difficile, è possibile **confrontare i risultati con i propri** e procedere.

Questo, a sua volta, ha reso necessaria l'introduzione di un'altra caratteristica innovativa: la **valutazione.** L'operato dell'utente, infatti, viene **valutato** e il voto è espresso, al termine di ogni esercizio, in trentesimi. La valutazione, sebbene puramente indicativa, ha lo scopo di fornire una **misura di miglioramento** (eseguendo lo stesso esercizio, naturalmente a meno errori corrisponde una valutazione più elevata) e allo stesso tempo di **disincentivare** il ricorso alla soluzione, in quanto questa comporta una grave penalità nel voto. In termini generali, la **valutazione** è calcolata effettuando una *media pesata* di singoli voti inseriti ad ogni passo del

problema, con un *peso* differente in base al tipo di domanda.

I test a risposta chiusa (i.e. il **test di ottimalità**) aggiungono un singolo voto, di peso 1, con il possibile valore di 30, corrispondente alla risposta esatta, oppure 0, in caso di risposta sbagliata. Alla selezione della variabile entrante, il voto candidato ad essere inserito parte dal valore 30 e si dimezza per ogni tentativo errato; così, se lo studente seleziona l'appropriata variabile entrante dopo due tentativi andati a vuoto, verrà aggiunto alla media un voto pari a 7.5, con peso 1.

I passi che prevedono l'inserimento di un *tableau* presentano un caso più complesso. Anche qui, lo studente è candidato a ricevere un voto pari a 30, con peso 3. Alla **convalida** del *tableau*, il voto viene diminuito di una quantità pari al numero di caselle con un valore errato, moltiplicato per $n/30$, dove n è il numero di caselle presenti in quel momento nel *tableau*. In questo modo, ogni casella ha un peso relativo alla dimensione del *tableau* attuale: è intuitivo pensare che sbagliare il valore di una casella, nel *tableau* di un problema a due vincoli e due variabili, sia più grave che sbagliare una casella in un problema artificiale a 7 vincoli e 7 variabili. Se si richiede la **soluzione** del problema, viene aggiunto uno 0 alla valutazione e la successiva **convalida**, *non* dà luogo all'aggiunta di un voto. Il peso elevato assicura, in primo luogo, che il ricorso alla soluzione non sia abusato; secondariamente, che la correttezza del *tableau* abbia un'importanza più elevata nell'esecuzione del problema.

§4.3.3 – Esportazione in AMPL

Un'ulteriore funzionalità innovativa disponibile in **Simply,** è l'**esportazione in linguaggio AMPL** (*A Mathematical Programming Language),* un linguaggio di programmazione d'alto livello, sviluppato dai **Bell Laboratories** e di proprietà della **AMPL Optimization, LLC**.

Lo scopo di **AMPL** è modellare problemi di ottimizzazione matematica, anche su larga scala. Può descrivere problemi di **Programmazione Lineare**, **nonlineare** e problemi di complementarietà, a valori continui o discreti. Utilizza una sintassi largamente simile al simbolismo matematico, il che permette una formulazione intuitiva per molte tipologie di problemi. I *file* di codice **AMPL** devono essere presi in *input* da altri programmi, chiamati collettivamente *solutori*, che si occupano di risolvere il problema e visualizzarne i risultati.

AMPL, assieme ad una selezione di solutori, è disponibile alla pagina

http://www.ampl.com/DOWNLOADS/index.html, in versione di prova limitata a 300 variabili e 300 vincoli, soggetta alle condizioni riportate alla pagina http://www.ampl.com/amplcond.html. Il programma **ampl.exe**, per piattaforma Windows, ha il compito di fornire i *file* di codice **AMPL** all'apposito solutore e visualizzarne i risultati.

Per effettuare l'esportazione, è sufficiente selezionare l'appropriato comando (cfr. **§4.5.1**) dall'interfaccia di **Simply**. Verranno creati i *file* **.mod** e **.dat**, da caricare nel programma **ampl.exe**.

§4.4 – Architettura e implementazione

In questo paragrafo verrà descritta l'architettura del *tool* sviluppato, sia **ad alto livello**, che in maniera più approfondita, analizzando le principali **classi Java**, con le loro relazioni e associazioni. Si metteranno inoltre in evidenza, per questo progetto, le potenzialità date dal linguaggio Java in termini di **programmazione ad oggetti** e **programmazione ad eventi** entrambe ampiamente utilizzate per lo sviluppo del *tool*.

I diagrammi delle classi, presentati all'interno di questo paragrafo, sono stati realizzati con l'ausilio del *plugin* **EclipseUML Studio**, sviluppato per Eclipse da **Omondo**, versione **3.4.1** del **novembre 2008**, scaricabile alla pagina http://www.uml2.org/download_studio_eclipse_3.4.html.

§4.4.1 – Principali passi implementati del Metodo del Simplesso

Nel **Capitolo 2** è stata descritta la procedura del **metodo del simplesso**. Tale procedura si presta ad essere divisa in *passi* (*step*) ben definiti, ognuno corrispondente ad una singola fase; di questa divisione è stato fatto uso per ordinare e semplificare l'implementazione dell'algoritmo.

L'esecuzione di quest'ultimo è **definita dall'utente**: il passaggio da uno stato all'altro avviene in modo chiaro, rigido e sempre per esplicita scelta. Questo permette al programma di mantenere un **flusso parallelo** di dati: un *set* interno, corrispondente ai valori attuali del problema (il *tableau*, l'insieme delle variabili di base, la riga e la colonna *pivot*) e un *set* esterno, corrispondente all'input fornito dallo studente.

Per ogni stato, il primo *set* viene preventivamente calcolato a partire dai dati disponibili; quando l'utente chiede il passaggio allo stato successivo, il *set* interno è confrontato con il *set* esterno e l'**esito** viene visualizzato.

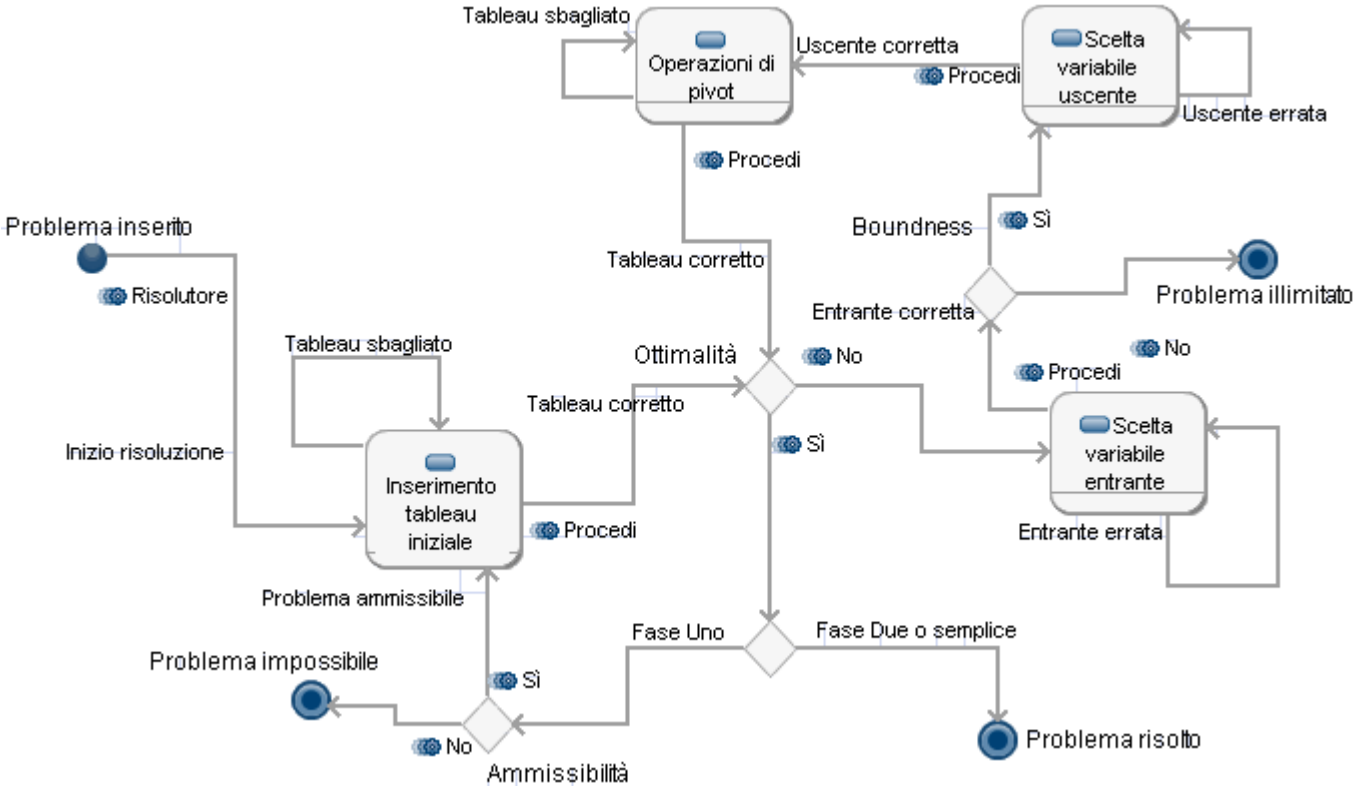


Fig. 19: Diagramma dell'esecuzione di un problema in Simply

Il flusso d'esecuzione di **Simply** può essere visualizzato attraverso un diagramma in cui ogni stato assume, come *input*, l'*output* dello stato precedente e l'*input* fornito dall'utente attraverso l'interfaccia del programma.

L'esecuzione interattiva ha inizio quando l'utente **immette un problema** nel meccanismo di risoluzione, che chiameremo **Risolutore**. In **Fig. 19**, lo stato denominato "Inserimento *tableau* iniziale" riassume le condizioni di **problema artificiale** o semplice: sono concettualmente le stesse, perché entrambe prevedono l'inserimento iniziale di dati nel *tableau*, con eventuale normalizzazione. L'azione indicata con "**Procedi**" rappresenta un esplicito comando dell'utente, tramite l'interfaccia grafica.

E' importante sottolineare che la correttezza o meno delle scelte dell'utente (sotto forma di valori numerici inseriti, o domande a risposta chiusa nel caso dei vari test) *non* influenza od ostacola il proseguimento dell'applicazione. La linea base del procedimento è data dal *set* elaborato internamente, che, per ipotesi, è corretto. L'*input* fornito dall'utente, per ogni

stato, ha solamente funzione di confronto ed è utilizzato per fornire **informazione** (sotto forma di messaggi testuali) e **valutazione**, entrambe come risultato di quest'ultimo.

Il nucleo della computazione è definito dagli stati "**Scelta della variabile entrante**", "**Scelta della variabile uscente**" e "**Operazioni di pivot**", che corrispondono alle omonime fasi del **metodo del simplesso**. Assieme a queste, si possono notare sia il **test di ottimalità** che il **test di *boundness***.

Tale ciclo si interrompe quando vengono raggiunte due delle possibili condizioni di fine: il problema è *illimitato*, il che causa la terminazione dell'algoritmo; oppure, il problema ha raggiunto l'*ottimo*. A questo punto, si hanno due ulteriori opzioni, determinate non dalla scelta dell'utente, ma dalla tipologia dei dati inseriti: o si è nella **Fase Uno** di un problema artificiale, oppure no, il che comprende l'essere nella **Fase Due**, o in un problema non artificiale. Nel caso di fine Fase Uno, si passa all'unico **test di ammissibilità**, che dà luogo alla terza e ultima possibile condizione di fine algoritmo (il problema è *inammissibile*), e successivamente, nel caso il test abbia successo, ad un nuovo inserimento del *tableau* iniziale. Nel secondo caso, invece, l'ottimalità diventa una **condizione di termine**.

L'esecuzione di un generico problema **artificiale** può essere suddivisa nei punti visualizzati in **Fig. 20**.

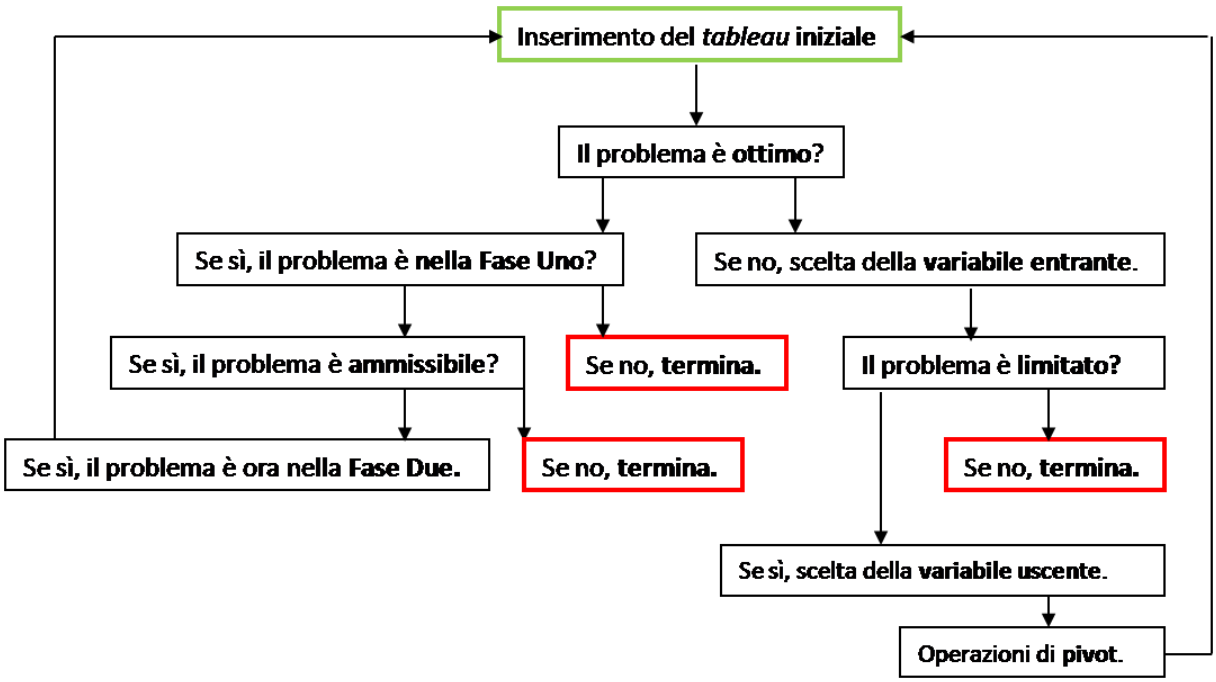


Fig. 20: Flusso dell'esecuzione di un problema artificiale in Simply

Si noti che la divisione, ad alto livello, è essenzialmente la stessa per un problema che richieda il **simplesso** o il **simplesso a due fasi**. In quest'ottica, possiamo considerare un problema che non richieda **forma artificiale**, equivalente ad un problema della Fase Due. Analizzando la letteratura, si può notare infatti come alcuni docenti preferiscano spiegare il **metodo del simplesso** direttamente in termini delle due fasi, anteponendo la spiegazione della Fase Due (un problema semplice) alla Fase Uno **[Roma et al., 2008]**.

Nel prossimo paragrafo, sarà spiegato il modo in cui le strutture sopracitate sono state trasposte in codice Java.

§4.4.2 – Architettura del progetto Simply

Nella stesura del codice, si è cercato il più possibile di realizzare una struttura che fosse allo stesso tempo robusta e semplice da mantenere e riutilizzare. La **struttura a oggetti** intrinseca del linguaggio Java si è prestata bene allo scopo. L'obiettivo è stato affrontato mantenendo, per quanto possibile, ogni classe per un solo scopo.

Lo schema in **Fig. 21** riassume le più importanti relazioni tra le classi del problema. Per evidenziare solo le connessioni principali, sono state rimosse molte delle associazioni e dipendenze secondarie. Inoltre, è stata omessa la **enum SyCo**, dato che tutte le altre classi vi si riferiscono come *repository* di costanti utilizzate nell'esecuzione del programma.

E' possibile notare i due principali canali di comunicazione tra le classi del progetto, rappresentate dalle associazioni tra le classi **OriginalProblemFrame** e **SolverFrame**, **SolverFrame** e **Solver**. Ognuna di queste classi istanzia o richiama uno e un solo oggetto della classe con cui si relaziona; tramite questo oggetto, indicato per entrambe le coppie nell'etichetta dell'associazione, avvengono tutti gli scambi di dati.

Sono inoltre evidenziate le classi ausiliarie **Coefficient**, utilizzata primariamente da **OriginalProblemFrame** e **SolverFrame**, e le classi **Matrix** e **Variable**, usate dalla classe **Solver.** Inoltre, le classi **Exporter**, **GraphicView** e **AboutBox** implementano funzionalità aggiuntive, rispettivamente l'esportazione in **AMPL**, il grafico del problema e la finestra di informazioni sul programma.

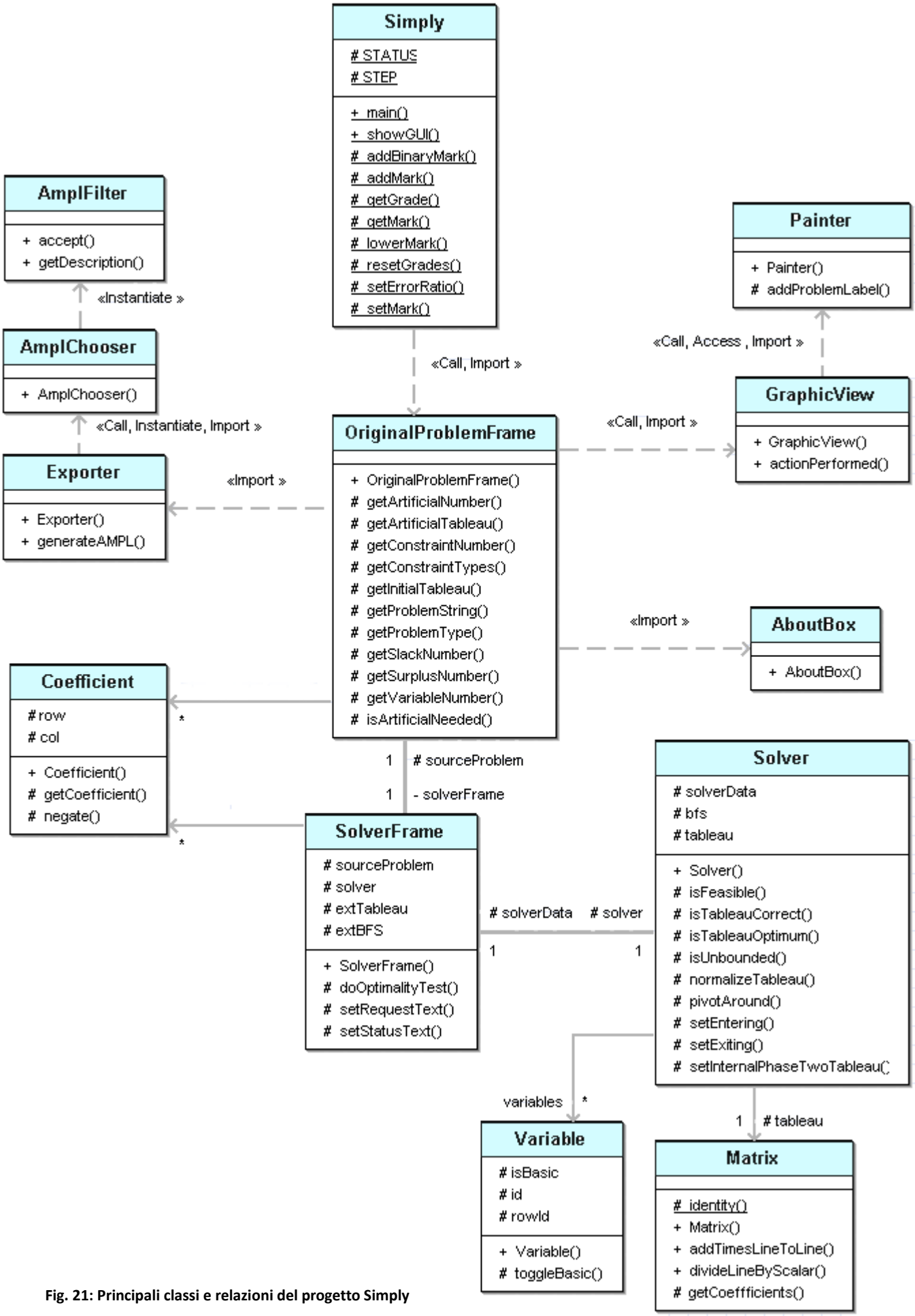


Fig. 21: Principali classi e relazioni del progetto Simply

Una delle strutture dati maggiormente utilizzate nel progetto è quella definita dalla classe **Matrix**, che rappresenta una matrice di valori *float* (a precisione singola). I principali metodi esposti dalla classe permettono:

- **Costruzione di una matrice** di dimensioni specificate, vuota oppure inizializzata ad un valore fornito in costruzione;
- **Scrittura e lettura** di singoli elementi in una matrice;
- **Divisione di una riga per uno scalare**, con modifica della matrice *in loco*;
- **Somma di una riga ad un'altra riga *n* volte**, con modifica della matrice *in loco*.

La seconda implementazione basilare in **Simply** è stata la **enum SyCo**, collezione di valori nominali, per codificare in maniera descrittiva e ordinata queste caratteristiche:

- **Stato del problema** (metodo semplice, artificiale della Fase Uno o Due, problema impossibile, illimitato o risolto)
- **Passo del problema** (inserimento del *tableau*, test di ottimalità, selezione entrante, ecc.)
- **Tipo di problema** (massimizzazione o minimizzazione)
- **Verso del vincolo** (maggioranza, minoranza o uguaglianza)

La classe primaria **Simply**, omonima al progetto, contiene il metodo **main()**, *entry point* del programma. Essendo il primo metodo eseguito, si occupa di istanziare e rendere visibile l'interfaccia grafica. Contiene anche i due campi statici **STATUS** e **STEP**, che registrano rispettivamente lo stato (globale) e il passo del problema, come elementi di **SyCo**; infine, implementa alcuni metodi statici per l'elaborazione della **valutazione**, utilizzati da ogni punto del problema. Si è scelto di implementarli come metodi statici, perché la valutazione è una sola nel corso del singolo esercizio e deve essere accessibile da ogni altra classe.

L'interfaccia grafica è stata assemblata utilizzando ed estendendo classi dei *package* grafici **java.awt** e **javax.swing**; le funzionalità offerte, sia in termini di oggetti che di gestione degli eventi, sono state necessarie per il completamento del progetto.

Le due principali classi dell'interfaccia sono **OriginalProblemFrame**, contenente la finestra principale, e **SolverFrame**, che definisce la finestra del risolutore. Entrambe ereditano dalla classe **javax.swing.JFrame**, una delle classi base per la creazione di una finestra.

Come classe ausiliaria alle due sopracitate, è presente la classe **Coefficient**, basata su **javax.swing.JFormattedTextField**. Quest'ultima permette in maniera nativa di ottenere un campo di testo, modificabile o meno, che risponda a precise regole, impostabili a piacimento, sulla formattazione (i.e. dei numeri inseriti). Per rappresentare i coefficienti del problema, includendo **funzione obiettivo, vincoli, termini noti** e più avanti il *tableu* stesso, è stato ritenuto utile utilizzare un formato numerico a virgola mobile, con una precisione di due cifre decimali. Oltre alle funzionalità basilari di **JFormattedTextField**, la classe **Coefficient** implementa le seguenti caratteristiche:

- **Controllo immediato dell'input** con *feedback* visivo in caso di errore (ad es. l'inserimento di un valore non numerico nel campo).
- ***Spatial awareness*** (conoscenza della propria posizione): ogni oggetto **Coefficient**, in costruzione del *tableau*, riceve un identificativo di riga e di colonna.
- **Lettura del valore** direttamente come *float*.
- **Negazione del valore** inserito nel coefficiente.

Tutti i campi di testo presenti sia nell'interfaccia di **OriginalProblemFrame** che in quella di **SolverFrame**, sono **array** mono- e bidimensionali di **Coefficient.**

La classe **OriginalProblemFrame** implementa molti metodi, la maggior parte dei quali volti al disegno dell'interfaccia o alla gestione degli eventi provenienti da essa. In particolare, sono state incluse le seguenti caratteristiche notevoli:

- Selettori a valori fissi, oggetti della classe **javax.swing.JSpinner**, che controllano rispettivamente il numero di variabili, il numero di vincoli e il tipo di problema. La modifica di uno dei selettori di variabili o vincoli, provoca una modifica visiva nel campo dei coefficienti: gli oggetti **Coefficient** che costituiscono i campi di testo della riga (o colonna) in questione, vengono resi visibili o invisibili, a seconda dei casi.

- Il metodo **getProblemString()**, che restituisce una **String** contenente il testo del problema, nella **forma standard** introdotta nel paragrafo **§1.2**, mediante un *parsing* dei valori nei campi **Coefficient**.

- Alcuni **metodi di lettura** per tutti i valori salienti del problema: numero di **variabili decisionali**, **slack**, **surplus** e **artificiali**, che restituiscono un intero; tipo di problema e vincoli, che restituiscono valori della **enum SyCo**.

- I metodi **getInitialTableau()** e **getArtificialTableau()** che, come si può facilmente intuire dai nomi, restituiscono un oggetto **Matrix** contenente i *tableau* iniziale per un problema semplice o per un problema artificiale della Fase Uno, con valori generati in base ai coefficienti inseriti.

L'utente, al momento di inserire un problema, utilizza l'interfaccia disegnata dalla classe **OriginalProblemFrame**, che si occupa quindi di conservare i dati originali per un riferimento futuro. Inoltre, l'azione di passaggio alla fase successiva, resa disponibile dal tasto **Risolutore**, crea un'istanza della classe **SolverFrame** opportunamente inizializzata.

SolverFrame accetta, nel suo costruttore, un oggetto della classe **OriginalProblemFrame**, utilizzato successivamente come riferimento per i dati iniziali lungo tutta l'esecuzione del programma. Anche quest'ultima classe, dovendo definire a sua volta un'interfaccia, contiene numerosi metodi per il disegno e il posizionamento degli oggetti interattivi, nonché per la definizione del loro comportamento.

Al momento della creazione dell'oggetto **SolverFrame**, esso crea un'istanza della classe **Solver**, che è il vero e proprio nucleo del programma. I *set* interno ed esterno di cui si parla nel paragrafo **§4.3.1**, sono qui implementati: un oggetto **Matrix** nella classe **Solver** è il *tableau* interno, un **array** bidimensionale di **Coefficient**, della stessa dimensione, è il *tableau* esterno. Analogamente, la **BFS** è un **array** di *float* nella classe "interna" ed un **array** di **Coefficient** nella classe "esterna".

Il seguente pseudocodice mostra il modo in cui le classi **Solver** e **SolverFrame** interagiscono per l'esecuzione di un generico passo del problema.

```
1:   corretto = false;
2:   calcola stato in Solver;
3:   if(conferma utente) {
4:        leggi valori da SolverFrame;
5:        confronta valori in Solver;
6:        if(confronto positivo)
7:             corretto = true;
8:        if(corretto)
9:             visualizza messaggio_ok;
10:       else
11:            visualizza messaggio_sbagliato;
12:  }
13:  if(corretto)
14:       STATO = prossimo_stato;
15:  else
16:       torna a riga 3;
```

Ad ogni passo, lo stato viene preventivamente calcolato e memorizzato nelle strutture dati in **Solver**. Quando l'utente chiede il passaggio allo stato successivo, i dati interni sono confrontati con quelli esterni, inseriti dall'utente stesso tramite l'interfaccia di **SolverFrame**. Se il confronto è positivo, il passaggio di stato è permesso; altrimenti, si richiede un nuovo confronto. In ogni caso, l'utente è informato dell'esito. Questo schema è costantemente applicato, sia nel caso di domande a risposta chiusa, come i vari **test** (i.e. ottimalità), sia nel caso si richieda l'inserimento di valori numerici.

Nella classe **SolverFrame**, il metodo **setNextStep()** si occupa di selezionare le azioni da eseguire a seconda dello *step* attuale. Altri metodi eseguono azioni più specifiche, come ad esempio la visualizzazione delle finestre di dialogo per le domande a risposta chiusa, oppure la risistemazione del *tableau* esterno per l'inizio della Fase Due. I metodi **setRequestText()** e **setStatusText()** si occupano di visualizzare i vari messaggi di stato e messaggi informativi nelle apposite sezioni dell'interfaccia di **SolverFrame**. A tale scopo, è stato mantenuto un archivio di oggetti **String** nella **enum SyCo**, ognuno contenente un messaggio da visualizzare, formattato in **HTML**

La classe **Solver** fa uso di una classe ausiliaria, **Variable**, che rappresenta una **variabile** del

problema. Un oggetto **Variable** possiede:

- Un campo *float* **c**, che rappresenta il suo *coefficiente* (da non confondersi con il suo *valore*);
- Un campo *int* **id**, che rappresenta l'indice della variabile ed è utilizzato per assegnare la posizione nella **BFS**;
- Un campo *int* **rowId** che rappresenta l'indice della **riga**, nel *tableau*, che contiene il valore di una variabile quando quest'ultima è **in base** (e che, quando la variabile è fuori base, è impostato a -1);
- Variabili *boolean* per stabilire se la variabile è **di base**, **slack**, **surplus** o **artificiale**;
- Metodi per la lettura e scrittura del campo **c** e per alternare lo stato di **base/fuori base**.

Tutte le variabili del problema, decisionali oppure no, sono mantenute in un **array** monodimensionale di oggetti **Variable**, creato in fase di inizializzazione dell'oggetto **Solver**.

La sincronia tra i due *tableau* e le due **BFS**, quando necessaria, è mantenuta dai metodi **getCoefficientTableau()** e **getCoefficientBFS()** della classe **Solver**, che si occupano di trasferire i valori del *tableau* interno in quello esterno, effettivamente rendendo visibili all'utente i valori del *tableau* interno. Quest'operazione accade solo quando l'utente chiede la **Soluzione** per il passo attuale, con l'apposito pulsante nell'interfaccia di **SolverFrame**.

Oltre a questi, i metodi principali della classe **Solver** comprendono:

- Il metodo **getBFS()**, per il calcolo della **BFS** interna a partire dal vettore di oggetti **Variable**;
- Metodi che restituiscono un valore *boolean*, per il *testing* della correttezza del *tableau* esterno contro il *tableau* interno e per i **test di ottimalità**, **ammissibilità** e ***boundness***;
- Il metodo **pivotAround()** che prende in ingresso due oggetti **Variable** ed esegue (sul *tableau* interno solamente) le **operazioni di pivot**, sfruttando i metodi sopracitati

della classe **Matrix**;

- Il metodo **normalizeTableau()** che esamina la prima riga del *tableau* interno e lo *normalizza* (cfr. paragrafo **§2.6**) se necessario;

- I metodi **setEntering()** e **setExiting()** per impostare, rispettivamente, la colonna della variabile *entrante* e la riga della variabile *uscente*;

- Il metodo **setInternalPhaseTwoTableau()** che si occupa di modificare il *tableau* interno, eliminando le **variabili artificiali**, per il passaggio alla Fase Due; effettivamente, creando un nuovo oggetto **Matrix** basato sul precedente.

Le classi **Exporter** e **GraphicView** implementano funzionalità accessibili dall'interfaccia di **OriginalProblemFrame**. In particolare, **Exporter** definisce i metodi per **esportare il problema in linguaggio AMPL**; le sue classi accessorie **AmplChooser** e **AmplFilter** , rispettivamente estensione di **javax.swing.FileChooser** e **javax.swing.filechooser.FileFilter**, permettono di visualizzare e selezionare i soli *file* rilevanti per **AMPL** in una finestra di dialogo per la selezione, dall'aspetto conforme al sistema operativo sottostante.

La classe **GraphicView**, a sua volta, definisce la finestra contenente il **grafico del problema** (per problemi di due variabili) e la sua classe accessoria **Painter**, estensione di **javax.swing.JComponent**, ridefinisce il metodo **paint()** della classe "madre" per visualizzare il grafico vero e proprio, attraverso l'uso della classe **javax.swing.Graphics2D**.

Infine, la classe **AboutBox** definisce una finestra di dialogo con informazioni sul programma e l'autore, accessibile dai menu **Aiuto** della finestra principale e del risolutore.

E' stata prestata particolare attenzione perché non si possano verificare casi di impossibilità a procedere, ad es. per la mancata esecuzione di uno dei metodi o per risultati inaspettati; ogni situazione testata dà luogo ad un **feedback** appropriato da parte dell'interfaccia grafica, in modo che l'utente sappia sempre cosa fare, come procedere o, se questo è il caso, che l'algoritmo è terminato.

§4.5 – Interfaccia e funzionalità

Nei paragrafi seguenti verranno esaminati i componenti principali dell'**interfaccia**, o **GUI** (*Graphical User Interface*) del programma e le funzionalità offerte, nel dettaglio. Particolare attenzione è stata posta sull'*usabilità* dell'interfaccia. L'enfasi è stata posta sulla *chiarezza* e la *modalità* dell'interfaccia, dove per modalità si intende l'informazione sullo *stato* del programma e della **GUI**, data in ogni istante. Ogni azione *significativa* dell'utente, spiegata, in ogni momento, da messaggi presentati sull'interfaccia, corrisponde ad una reazione appropriata della stessa.

§4.5.1 – Finestra principale

La finestra principale del programma contiene sia le opzioni primarie utili all'utente per avviare un esercizio, che brevi informazioni sul funzionamento dell'interfaccia.

La **Fig. 22** mostra l'aspetto della finestra all'avvio del programma. In alto a sinistra, un pannello testuale fornisce le **informazioni basilari** su come procedere. A destra, tre selettori permettono il controllo sui **parametri principali** del problema: numero di variabili, numero di vincoli e tipologia dell'ottimizzazione (massimo o minimo). I selettori sono **non editabili**: i valori sono predefiniti e accessibili tramite gli appositi pulsanti (o frecce). In questo modo, si limita di molto la possibilità di errore (ad es. inserire un valore oltre il massimo numero previsto, oppure non numerico), anche per il selettore dell'ottimizzazione.

Il pannello dei **coefficienti** consente di inserire un problema. I campi sono disposti in modo da simulare un sistema di disequazioni ed equazioni; per mantenere l'interfaccia quanto più possibile regolare ed ordinata, si è deciso di non ripetere le variabili in ogni equazione, ma semplicemente in cima alla colonna che corrisponde loro. Per ogni vincolo, un selettore permette di scegliere un verso: maggioranza, minoranza od uguaglianza.

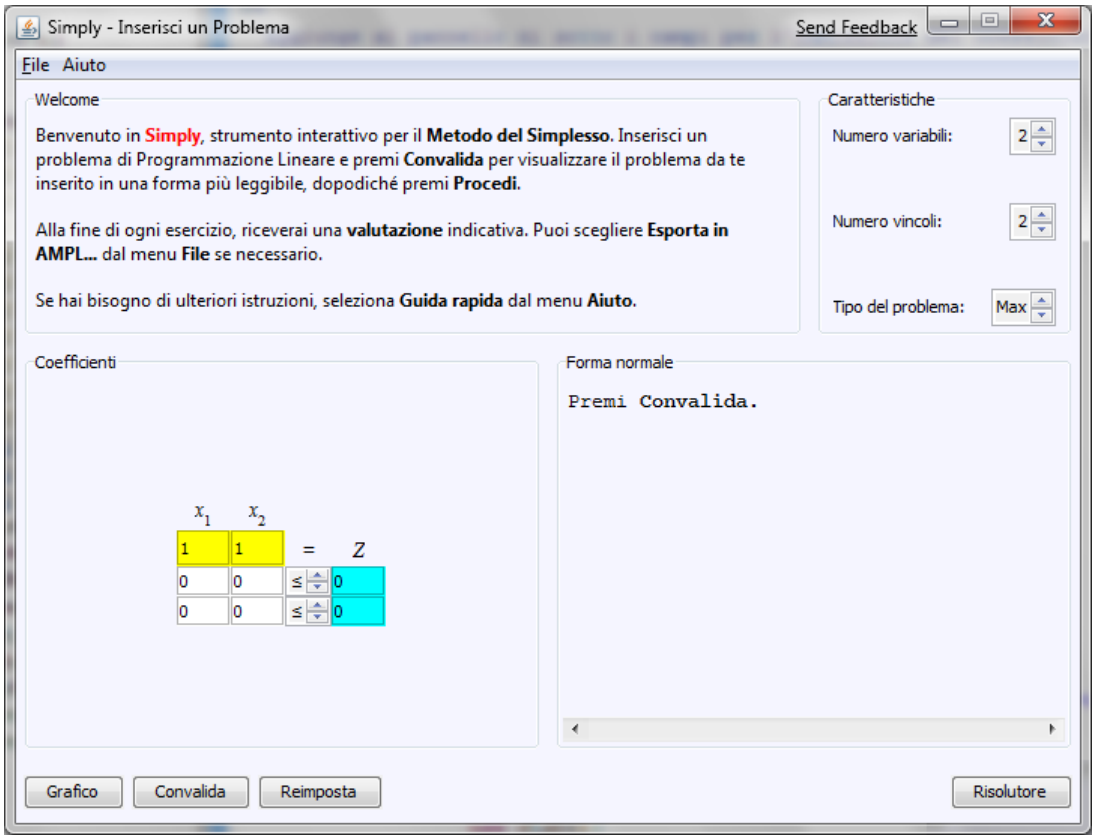


Fig. 22: Finestra principale di Simply.

Un **cambiamento nelle dimensioni** del problema si riflette immediatamente sul pannello dei coefficienti: se l'utente scorre avanti o indietro i selettori delle variabili o dei vincoli, vengono aggiunte righe o colonne in maniera appropriata. I campi aggiuntivi vengono aggiunti sempre in ultima posizione, ossia nuove colonne (corrispondenti a nuove variabili) a destra e nuove righe (corrispondenti a nuovi vincoli) in basso.

I campi numerici presentano una varietà di caratteristiche volte ad aiutare l'utente. Per prima cosa, sono **preformattati** con un massimo di due cifre decimali e il punto come separatore; si è pensato che una maggiore precisione, fosse al di fuori degli scopi del programma. Se lo studente inserisce un numero decimale con più di due cifre dopo la virgola, questo sarà arrotondato a due cifre appena il campo perde focalizzazione. Inoltre, i campi sono **validati** ogni volta che viene inserito, modificato o cancellato un carattere: se l'utente inserisce un carattere non numerico, o non interpretabile come numero, il campo appare immediatamente circondato da un bordo rosso, a segnalare l'errore. (I caratteri "f" e "d", come suffisso, indicano rispettivamente che il numero precedente è a precisione singola o doppia, perciò non sono segnalati come errore). Il bordo sparisce se l'utente corregge l'errore, immettendo un numero o cancellando il carattere non interpretabile; se l'utente si

limita a spostarsi su un altro componente dell'interfaccia, il campo ritorna all'ultimo valore valido inserito.

Dopo aver inserito un problema, comprensivo degli zeri dove necessario (un campo non può mai rimanere vuoto, perché il campo vuoto viene considerato errore), l'utente è invitato a premere il tasto **Convalida** (in basso a sinistra) per visualizzare, nel pannello centrale a destra, il problema in una forma più leggibile. Questo passo, mostrando una forma pressoché universalmente riconosciuta per i problemi di **Programmazione Lineare**, serve per permettere all'utente di rendersi conto se abbia inserito il problema desiderato, senza sviste. La **Fig. 23** mostra questo meccanismo in azione. Se si lasciano tutti i coefficienti delle variabili decisionali, presenti in un vincolo, pari a 0, il vincolo stesso non viene visualizzato (in quanto, effettivamente, non esiste). Per problemi di grandi dimensioni, o con coefficienti elevati, le equazioni e disequazioni potrebbero superare la dimensione orizzontale del pannello; in questo caso, si attiva la barra di scorrimento presente in calce al pannello. E' necessario effettuare la **Convalida** del problema almeno una volta, prima di procedere alla fase successiva.

I tasti di comando dell'interfaccia sono situati nella parte inferiore della finestra. Il tasto **Grafico**, attivo solo se il numero di variabili è impostato a 2 o meno (diventa accessibile o inaccessibile a seconda del valore sul selettore delle variabili), permette di accedere ad una visualizzazione grafica del problema. Il tasto **Reimposta** azzera i campi di inserimento dei coefficienti e riporta tutti i vincoli ad essere di minoranza: utile se si desidera inserire un nuovo problema e non si vuole cancellare manualmente i singoli campi. Infine, nell'angolo in basso a destra, il tasto **Risolutore** avvia l'ambiente di risoluzione del problema.

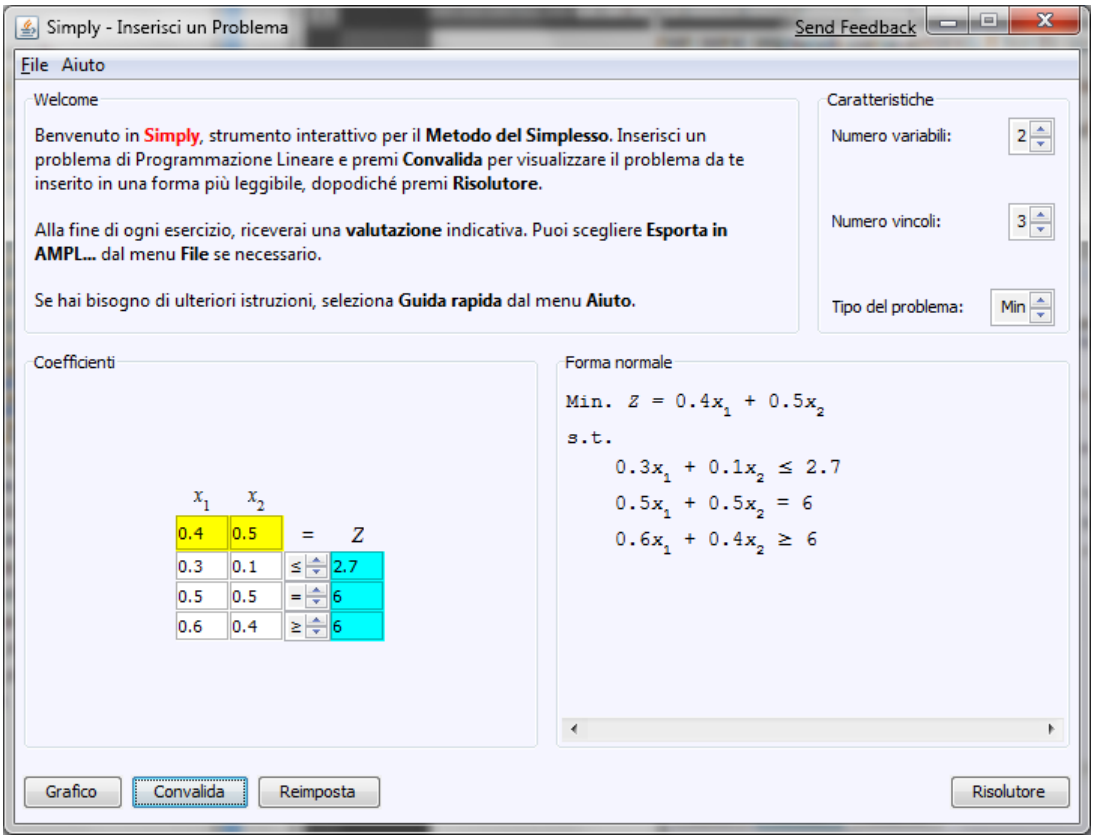


Fig. 23: Inserimento e convalida di un problema

Il menu **File** contiene due comandi: **Esporta in AMPL** e **Esci**. Quest'ultimo, attivabile in ogni momento. chiude il programma. Il primo comando permette l'esportazione del problema inserito in linguaggio **AMPL;** in particolare, fa comparire una finestra di selezione *file*. E' sufficiente digitare un nome, senza estensioni (verrebbero comunque scartate), per creare i documenti rilevanti per **AMPL** (cfr. **§4.3.3**). Se il nome contiene caratteri illegali per un *file*, oppure non si possiedono i permessi per creare un file nella posizione scelta, un messaggio di errore informa l'utente dell'accaduto; se si seleziona un file già esistente, all'utente viene chiesto di confermare la sovrascrittura. In caso di successo, un messaggio conferma la creazione dei file riportando nome e percorso.

Infine, il menu **Aiuto** contiene il comando **Guida rapida**, che visualizza un pannello informativo con una guida all'uso funzionalità offerte dalla finestra principale.

§4.5.2 – Grafico

La visualizzazione grafica del problema, ottenibile (per problemi a una o due variabili) premendo il tasto **Grafico** dalla finestra principale, presenta il problema su un grafo cartesiano. Il tasto non viene disattivato quando un grafico è aperto:ciò permette allo

studente di confrontare più grafici per volta, ad esempio per visualizzare la regione ammissibile modificata al cambiamento di un vincolo.

I **vincoli** sono segnati in colore **blu**, i vincoli di **uguaglianza** in azzurro e la **regione ammissibile**, nello stesso tono. Il *fascio di rette* definito dalla funzione obiettivo è rappresentato con rette di colore **rosso** e tratteggiate, convenzionalmente poste all'incontro tra i vincoli e l'asse delle ascisse.

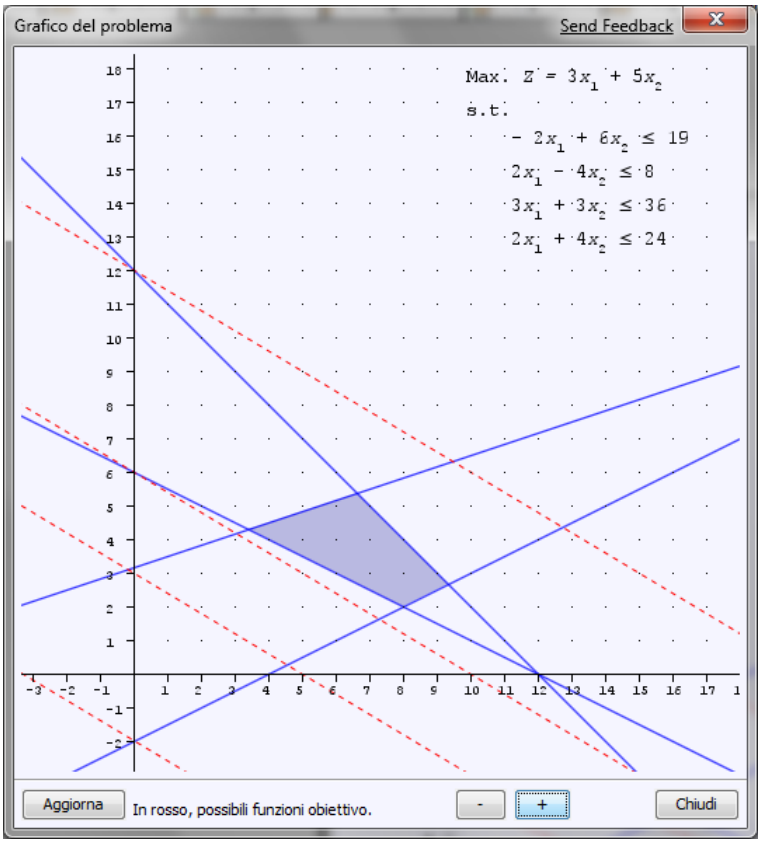


Fig. 24: Grafico di un problema a due variabili e quattro vincoli.

Il piano cartesiano è centrato sui valori del primo quadrante, perché il programma suppone che tutte le variabili siano maggiori o uguali a 0; i valori al di fuori del primo quadrante, perciò, non sono rappresentati. La **Fig. 24** rappresenta un problema con quattro vincoli, due di minoranza e due di maggioranza, con evidenziata la regione ammissibile risultante. In alto a destra, è rappresentato il problema in una **forma facilmente leggibile**, la stessa del pannello centrale destro della finestra principale, per rendere più chiaramente visibile quale problema si stia rappresentando.

La **funzione obiettivo**, nelle sue possibili rappresentazioni al variare di Z, è un indice visuale per aiutare la ricerca dell'ottimo per via grafica. Non viene fornito l'ottimo, né alcun valore

numerico: l'idea è che lo studente consideri il grafico rappresentato come una guida e trovi il valore ottimo in autonomia, completando l'algoritmo o ridisegnando il problema su carta.

E' possibile, tenendo aperta una o più visualizzazioni grafiche, modificare dei campi nella finestra principale e premere il pulsante **Aggiorna** per ridisegnare il grafico. L'aggiornamento coinvolge solo la finestra su cui si preme il pulsante, per cui più grafici possono rappresentare diversi problemi.

I pulsanti **+** e **–** permettono di aumentare o diminuire il fattore di scala del grafico, per problemi che richiedono una maggiore precisione o una più ampia visuale. Siccome i valori segnati sugli assi possono diventare difficili da leggere, a livelli bassi di zoom vengono visualizzati solo i valori *pari* della scala; la griglia rimane invece invariata. L'ingrandimento è centrato sul punto (0, 0) che, perciò, rimane sempre nella stessa posizione.

Infine, il pulsante **Chiudi** permette di congedare la finestra.

§4.5.3 – Risolutore

In questo paragrafo verrà proposta passo per passo la risoluzione del problema inserito in **Fig. 23**, un problema risolvibile con il **metodo a due fasi**, che pertanto consente di visualizzare la totalità delle caratteristiche dell'interfaccia di risoluzione.

La pressione del pulsante **Risolutore**, nella finestra principale, dà l'avvio alla sessione di risoluzione del problema. Se lo studente preme il pulsante lasciando valori non numerici in uno dei campi, **Simply** assume automaticamente l'ultimo valore valido inserito (si ricorda che i campi numerici *validano* ad ogni azione che origina da loro, compresa la perdita della focalizzazione).

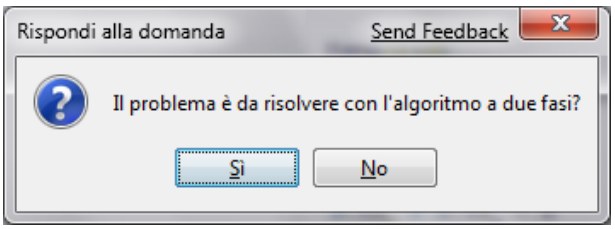


Fig. 25: Domanda sul problema artificiale

Se il programma rileva termini noti negativi, viene visualizzato un messaggio che ricorda allo studente di tenere conto di questo, nello stabilire se il problema deve, oppure no, essere risolto con il **metodo artificiale**. Dopodiché, compare il messaggio in **Fig. 25.** A seconda della

correttezza della risposta, si avrà un messaggio di conferma o di errore, con brevi indicazioni (non esaustive) sul motivo. Si tratta solo del primo di una serie di messaggi a risposta chiusa, con lo scopo di guidare e far riflettere lo studente sulle varie fasi del **metodo del simplesso**.

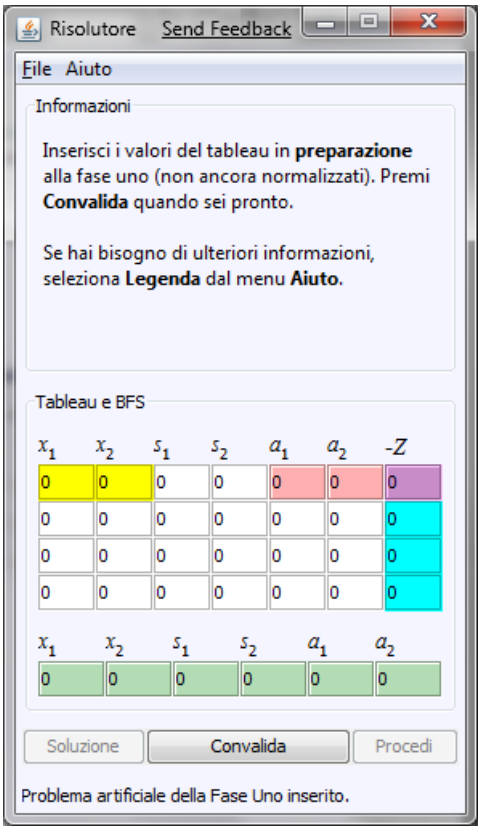


Fig. 26: Iniziale presentazione del risolutore

Successivamente, compare l'interfaccia vera e propria del risolutore, rappresentata in **Fig. 26**. Nella metà superiore, sono presentate le **istruzioni del passo attuale**. Sebbene le possibilità di errore siano minimizzate da accorgimenti sul comportamento dell'interfaccia, è importante che lo studente segua con precisione le indicazioni presenti in questo pannello, che è provvisto col preciso scopo di *guidare* all'uso dell'interfaccia. Nella metà inferiore della finestra, risiedono i campi numerici del *tableau* e della **BFS** attuale. Infine, alla base della finestra, sono visibili i tre tasti di comando e una barra di stato.

Il primo passo è l'**introduzione del *tableau* iniziale**. Il *tableau* ha la forma presentata in **[Hillier & Lieberman, 2005]**, con la funzione obiettivo in alto e i termini noti a destra, per un problema di *massimizzazione*. Questo, come si nota in figura, ha il risultato di invertire i segni della funzione obiettivo in caso di problemi di *minimizzazione*. I campi numerici presentano le stesse caratteristiche di quelli nella finestra principale: validazione dell'input, formato a due cifre decimali con il punto come separatore, segnalazione di valore impossibile da

interpretare.

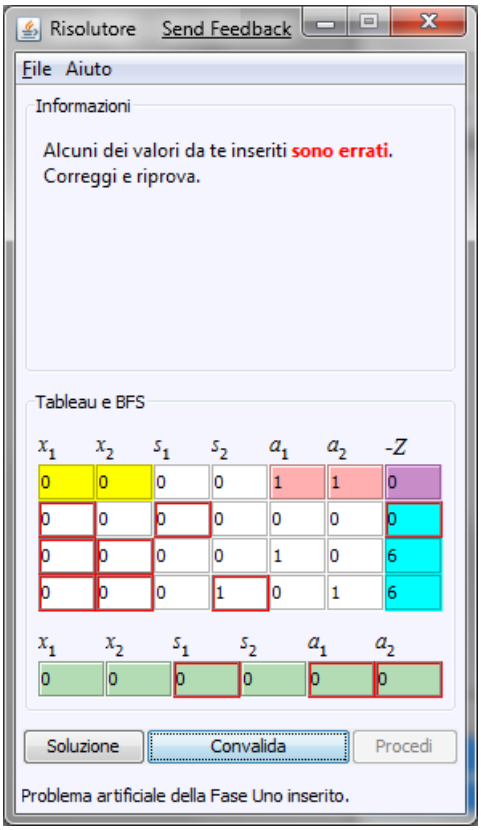


Fig. 27: Risultato di una convalida parzialmente errata.

Dopo aver introdotto i valori desiderati, è necessario premere il tasto **Convalida**, in basso e al centro. Questo tasto, qui e in tutto il resto del problema, causa il confronto tra il *tableau* inserito e quello internamente calcolato dall'algoritmo: a seconda della correttezza o meno del primo, viene visualizzato nella parte alta della finestra un messaggio di conferma, oppure di errore. In quest'ultimo caso, i campi errati sono evidenziati in rosso (vedi **Fig. 27**). Ciò permette allo studente di **identificare prontamente il proprio errore** e dedicarsi alla correzione. In caso di errore, viene attivato il tasto **Soluzione**: alla pressione di questo tasto, il *tableau* è compilato con i valori corretti per il passo attuale. L'area dei messaggi, a questo punto, visualizza una notifica che scoraggia l'uso della soluzione; inoltre, un voto nullo viene aggiunto alla media delle valutazioni, per penalizzare ulteriormente l'uso di questa possibilità. Il tasto viene disattivato dopo ogni utilizzo e riattivato solo in caso di nuovo errore alla prossima **Convalida** del problema.

Se la convalida non dà errori, il tasto corrispondente si disattiva e, invece, si attiva il tasto **Procedi**. La pressione di quest'ultimo porta alla fase di **normalizzazione del *tableau*** per la Fase Uno, opportunamente segnalata nell'area dei messaggi. Questo passo è analogo al

precedente: è necessario inserire i valori esatti nel *tableau* (dopo averli calcolati in autonomia) e **Convalidare** il risultato, premendo il tasto **Procedi** quando possibile. Il *design* dell'interfaccia fa sì che questo succeda solo quando l'intero *tableau* è corretto, per evitare incongruenze.

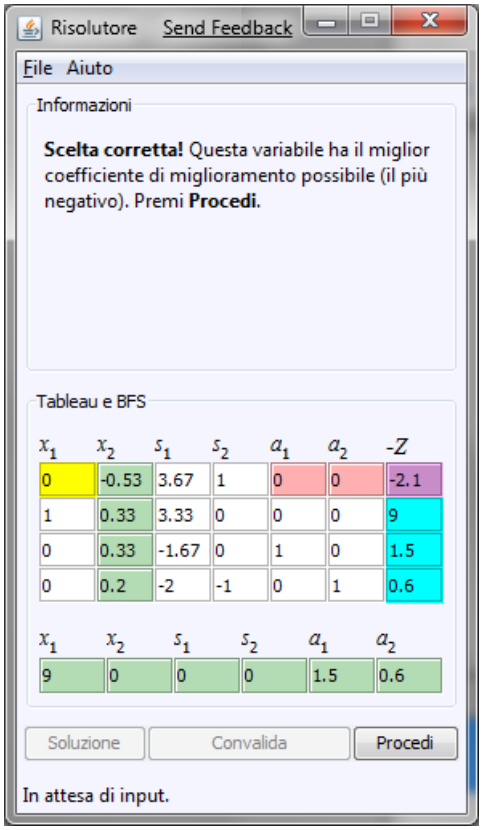


Fig. 28: Selezione della variabile entrante

Prima di passare oltre, viene presentato un messaggio che chiede se il *tableau* inserito è **ottimo** oppure no. Si tratta del **test di ottimalità**, che come la maggior parte delle altre fasi del **simplesso**, è richiesto interattivamente. In modo analogo alla domanda preliminare sull'artificialità, lo studente può ottenere un messaggio di conferma o di errore, a seconda della risposta, con una breve spiegazione. Il **test di ottimalità** è presente ogni volta che si inserisce e convalida un nuovo *tableau*, tranne nei due casi (l'inizio della Fase Uno, come si è visto, e l'inizio della Fase Due) in cui è presentato solo dopo la normalizzazione del *tableau*.

La fase successiva è la **selezione della variabile entrante**. Come segnalato nell'area messaggi, lo studente deve scegliere il **primo elemento della colonna pivot** (nel caso della **Fig. 28**, la casella con il valore -0.53). Solo la prima riga dei campi numerici, e solo in questa fase del problema (ad ogni sua ripetizione), risponde a questo tipo di input. La colonna scelta viene evidenziata in colore e, nell'area messaggi, compaiono notifiche sulla correttezza o meno

della scelta. Durante questa fase, tutti i pulsanti sono disabilitati, per limitare le possibilità di errori e incongruenze nella procedura; è possibile solamente selezionare colonne, fino alla selezione della colonna corretta. A questo punto, i campi numerici della prima riga non hanno più la possibilità di essere selezionati (per evitare confusioni) e così, la **colonna pivot** rimane correttamente evidenziata per le fasi successive. Il tasto **Procedi** si riattiva.

All'ulteriore pressione di quest'ultimo, compare una domanda chiusa sulla ***boundness*** (la condizione per cui la **regione ammissibile** è limitata, garantendo così l'esistenza di soluzioni) del problema; di nuovo, analogamente alle altre domande a risposta chiusa, l'utente riceverà un messaggio di conferma o di errore. Il **test di *boundness*** è presente dopo ogni selezione della variabile entrante; si è deciso di inserirlo a questo punto, perché prevede un controllo sulla **colonna pivot** appena selezionata. Alla risposta, si disattiva il tasto **Procedi**.

Il punto successivo è la **selezione della variabile uscente**. Tale scelta è praticata in maniera analoga alla fase precedente; viene effettuata tramite clic sulla **prima casella della riga pivot**. L'area dei messaggi presenta ora un'informazione appropriata, compreso un promemoria sull'uso del **test del minimo rapporto**. Come sopra, solo la prima colonna dei campi numerici, escludendo l'elemento nella prima riga, risponde al clic con la selezione; e solo in questa fase del problema.

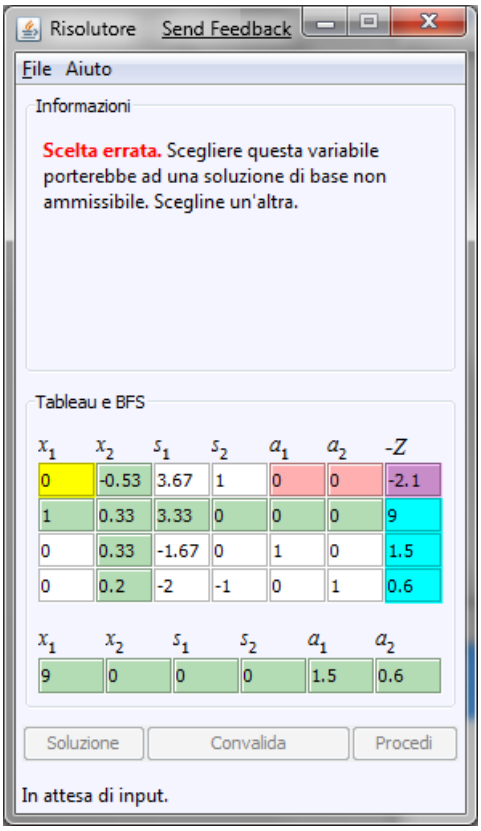


Fig. 29: Selezione della variabile uscente

In **Fig. 29** è rappresentata una possibile selezione, in questo caso errata. Si noti che, sia in questa fase che nella precedente, nel caso di parità tra le possibili scelte viene applicata la **Regola di Bland [Roma et al., 2008]**: è scelta la variabile con indice minore. Anche qui, come prima, è possibile selezionare righe diverse fino alla scelta dell'esatta **riga pivot**: a quel punto, la prima colonna perde la capacità di rispondere ai clic e si riattiva il tasto **Procedi**.

Il passo successivo consiste nelle **operazioni di pivot**. Allo studente è richiesto di modificare il *tableau*, tenendo conto delle nuove **variabili di base**. La **riga e la colonna pivot** rimangono evidenziate, per rendere immediatamente visibile quali siano. Si applica l'usuale procedura di **Convalida**, **Soluzione** (se desiderato) e possibilità di **Procedere** solo a convalida avvenuta con successo. Un nuovo **test di ottimalità** conclude il ciclo, che si ripete in tutte le sue parti fino al raggiungimento dell'ottimo.

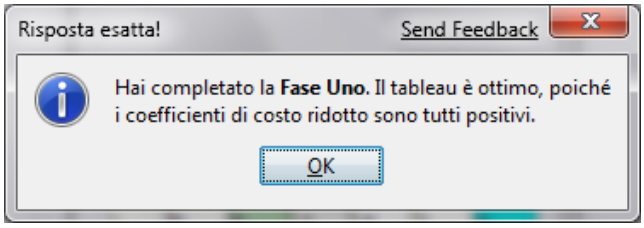


Fig. 30: Raggiungimento dell'ottimo alla fine della Fase Uno

Quando ciò avviene, un messaggio, visualizzato in **Fig. 30**, informa lo studente della fine della Fase Uno. A questo punto si trova l'unico **test di ammissibilità** del problema, ancora sotto forma di domanda a risposte chiuse.

Esaurito il test, viene presentato il *tableau* di preparazione alla Fase Due, con le **variabili artificiali** già rimosse (vedi **Fig. 31**). Questa è l'*unica* procedura automatizzata del problema, per ragioni d'implementazione.

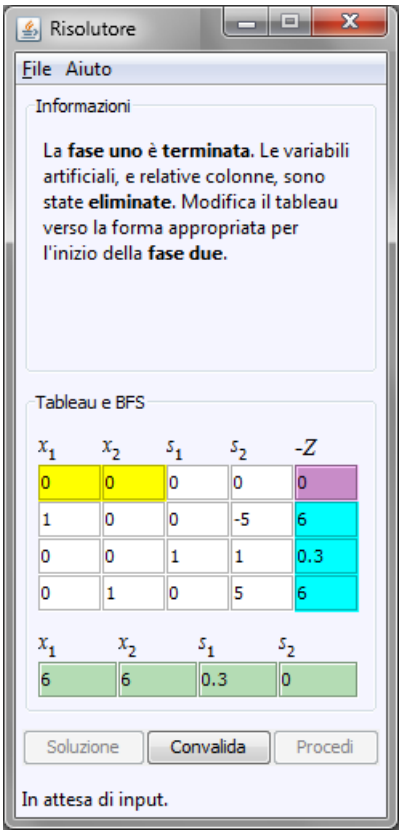


Fig. 31: Termine della Fase Uno

A questo punto, viene chiesto di inserire i valori appropriati (in questo caso, i coefficienti della funzione obiettivo) e normalizzare il *tableau*: questa è la seconda ed ultima occasione in cui ad una fase di **Convalida** non segue un **test di ottimalità**, che tuttavia è presente dopo la normalizzazione. Da questo punto, i passi e le possibilità sono esattamente gli stessi delle precedenti iterazioni, fino al raggiungimento dell'ottimo.

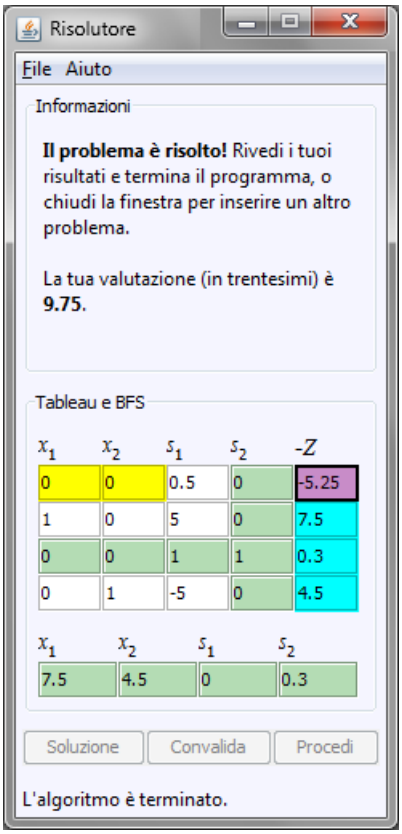


Fig. 32: Raggiungimento dell'ottimo

Alla fine del procedimento, viene comunicata allo studente la **valutazione**, calcolata in base alle scelte fatte (cfr. **§4.3.2**) e, inoltre, l'ottimo è evidenziato mediante un bordo nero marcato sulla casella, visibile in **Fig. 32**.

Capitolo 5

Test del software

In questo capitolo verranno presentati i test che completano lo sviluppo del software **Simply**. Saranno eseguiti alcuni problemi, di difficoltà crescente, che vadano a toccare tutti i possibili percorsi definiti dal codice, come raccomandato in letteratura **[McConnell, 2004]**.

§5.1 – Test effettuati e strumentazione utilizzata

All'inizio della fase di validazione e verifica del software implementato, si sono delineati i seguenti casi di test:

- **Caso 1:** problema banale, con due variabili e tre vincoli, a coefficienti interi e validazione del grafico **[Hillier & Lieberman, 2005]**;
- **Caso 2:** Un problema semplice, che necessiti del **metodo a due fasi**, con due variabili e tre vincoli, a coefficienti decimali, e validazione del grafico ottenuto **[Hillier & Lieberman, 2005]**;
- **Caso 3:** Un problema risolvibile con il **metodo a due fasi**, che presenti una **variabile artificiale** in base, pari a 0, alla fine della Fase Uno **[Archetti et al., 1989]**;
- **Caso 4:** Un problema illimitato e uno impossibile, entrambi con riscontro visivo;
- **Caso 5:** Un problema a coefficienti casualmente generati, con 7 variabili e 7 vincoli, tutti di maggioranza, in modo da ottenere un *tableau* il più esteso possibile.

Per verificare il superamento di ciascuno dei casi di test, si utilizzerà la funzione **Esporta in AMPL** del programma **Simply**, e si confronteranno i risultati finali con quelli ottenuti caricando i *file* **.mod** e **.dat** nel programma **ampl.exe** ed eseguendo i comandi `solve` e `display x`. Ciò avrà l'ulteriore risultato di verificare la correttezza delle procedure di esportazione.

§5.2 – Caso di test n. 1: Problema banale

Il primo test consiste nell'inserimento di un problema elementare, appartenente ad una tipologia facile da incontrare all'inizio di un corso di Ricerca Operativa. Si riporta qui il problema nella sua **forma standard**:

$$\text{Max. } Z = 3x_1 + 5x_2$$
$$\text{s. t.}$$
$$x_1 \leq 4$$
$$x_2 \leq 12$$
$$3x_1 + 2x_2 \leq 18$$
$$x_{1\ldots2} \geq 0$$

Successivamente, il problema è stato inserito nell'interfaccia principale di **Simply**; ne è stato subito visualizzato il grafico, rappresentato in **Fig. 33**. La regione ammissibile evidenziata da **Simply** coincide con quella riportata in **[Hillier & Lieberman, 2005]**.

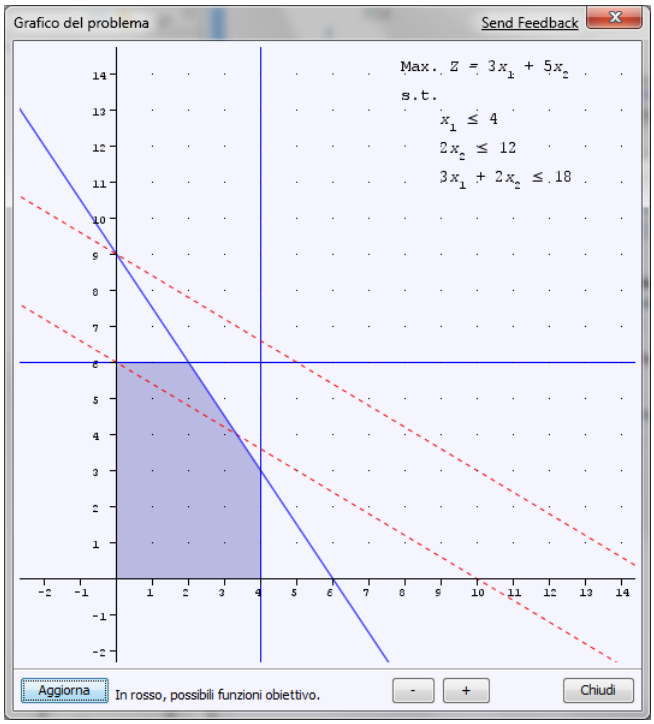


Fig. 33: Grafico del problema di test n. 1

Il terzo *step* della procedura di test consiste nell'**esportazione in AMPL** e al caricamento dei *file* nel programma **ampl.exe**.

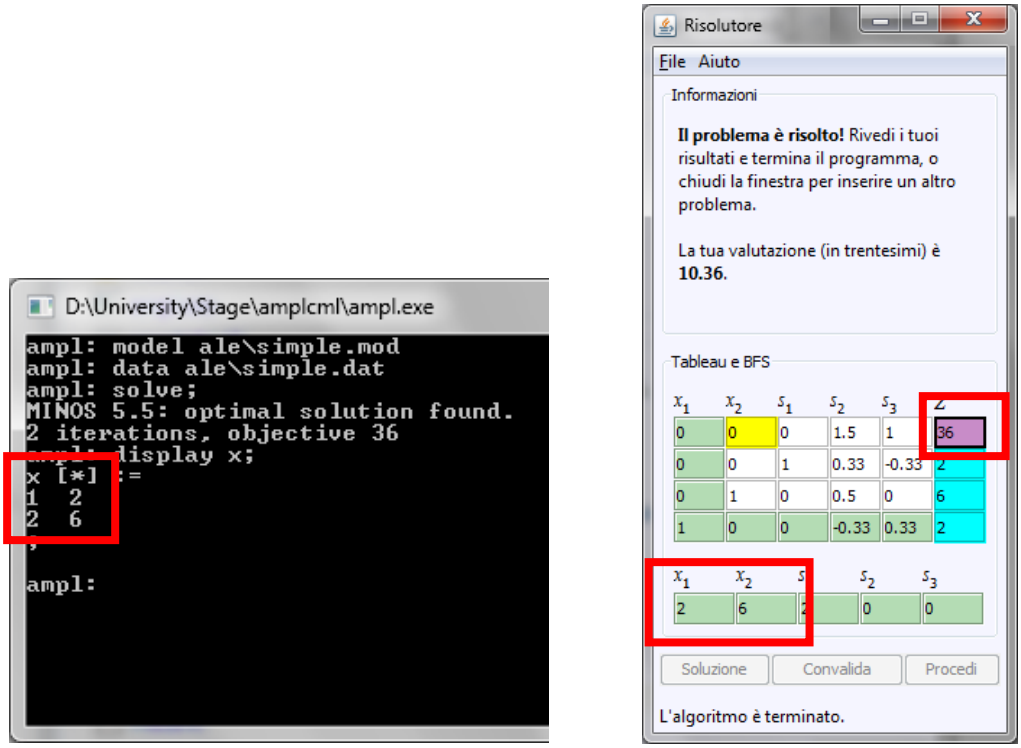


Fig. 34: Console di ampl.exe e schermata finale del risolutore per il problema di test n. 1

Come si può notare in **Fig. 34**, il valore ottimo è 36 e le due variabili, entrambe **in base** nella soluzione ottima, valgono rispettivamente 2 e 6. Tali risultati corrispondono con quelli ottenuti alla risoluzione del problema in **Simply**.

§5.3 – *Caso di test n. 2: Problema artificiale*

Il secondo test ha richiesto la soluzione di un problema risolvibile con il **simplesso a due fasi**. Anche questa tipologia può essere facilmente trovata in un corso di Ricerca Operativa, se non addirittura come parte di un esame. Riportiamo qui il testo del problema.

$$\text{Min. } Z = 0.4x_1 + 0.6x_2$$
$$\text{s.t.}$$
$$0.3x_1 + 0.1x_2 \leq 2.7$$
$$0.5x_1 + 0.5x_2 = 6$$
$$0.6x_1 + 0.4x_2 \geq 6$$
$$x_{1\ldots2} \geq 0$$

Anche stavolta, il grafico (**Fig. 35**) corrisponde con quanto riportato in **[Hillier & Lieberman, 2005]**. Si può notare il vincolo di uguaglianza, riportato nello stesso colore azzurro della regione ammissibile.

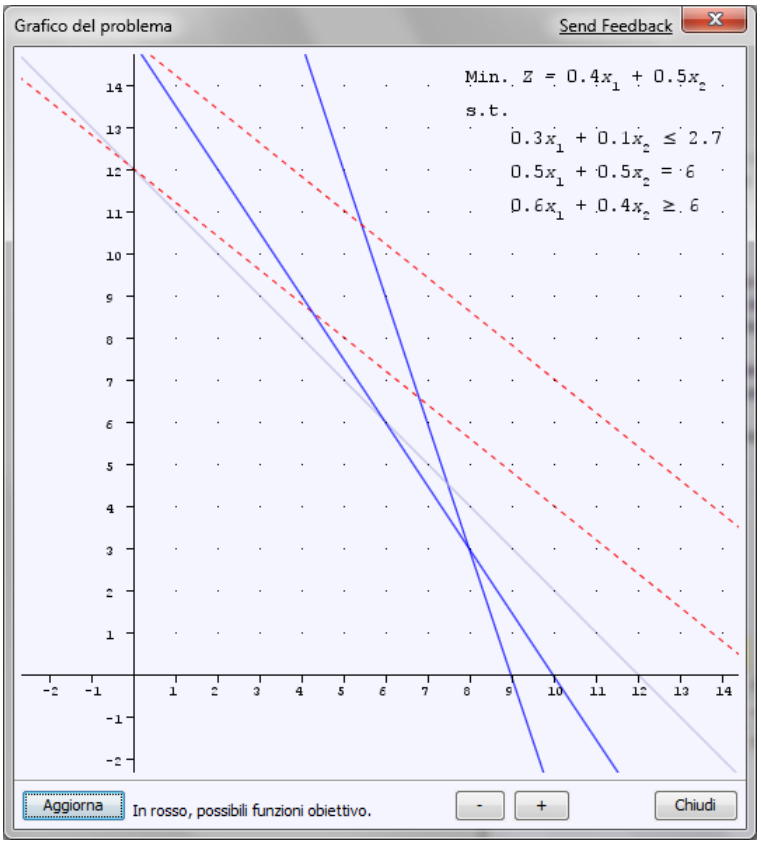


Fig. 35: Grafico del problema di test n. 2

Di seguito, in **Fig. 36**, è possibile confrontare l'*output* della console di **ampl.exe** per il

problema e la schermata finale del risolutore.

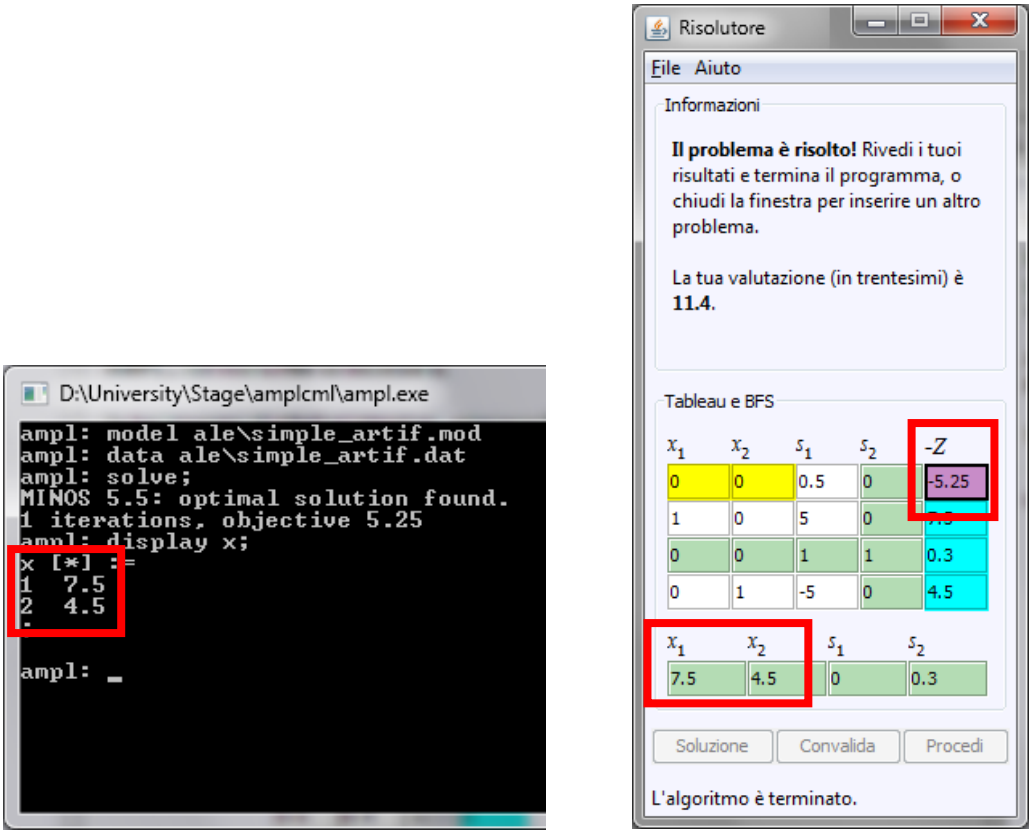


Fig. 36: Console di ampl.exe per il problema di test n. 2

Il valore ottimo è 5.25; le due **variabili decisionali** sono entrambe in base al raggiungimento della soluzione ottima, e hanno valore rispettivamente 7.5 e 4.5. Tali valori corrispondono, come evidenziato nella precedente figura. Si noti che **Simply**, se il problema è di *minimizzazione*, massimizza il valore di $-Z$; in questi casi, l'ottimo è l'inverso del valore rappresentato.

§5.4 – Caso di test n. 3: Problema artificiale con vincolo ridondante

Il terzo problema di test mostra un caso particolare, riportato in **[Archetti et al., 1989]**. La **forma standard** è qui di seguito riportata.

$$\text{Min. } Z = -x_1 + 2x_2 - 3x_3$$
$$\text{s.t.}$$
$$x_1 + x_2 + x_3 = 6$$
$$-x_1 + x_2 + 2x_3 = 4$$
$$2x_2 + 3x_3 = 10$$
$$x_3 \leq 2$$
$$x_{1\ldots2} \geq 0$$

Il problema conduce ad avere, alla fine della Fase Uno, **una soluzione di base degenere** in cui una variabile artificiale rimane in base, ma pari a 0; il che, a sua volta, significa che un vincolo è ridondante. L'algoritmo, in questo caso, può comunque continuare verso la Fase Due. La situazione alla fine della Fase Uno è rappresentata in **Fig. 37**.

Fig. 37: *Tableau* alla fine della Fase Uno per il problema di test n. 3

Il valore ottimo è -4 e le **variabili decisionali** hanno tutte valore pari a 2. Come si può

vedere in **Fig. 38**, il risultato corrisponde. Si noti anche che il terzo vincolo dall'alto è stato virtualmente eliminato dal *tableau* nel corso delle operazioni algebriche.

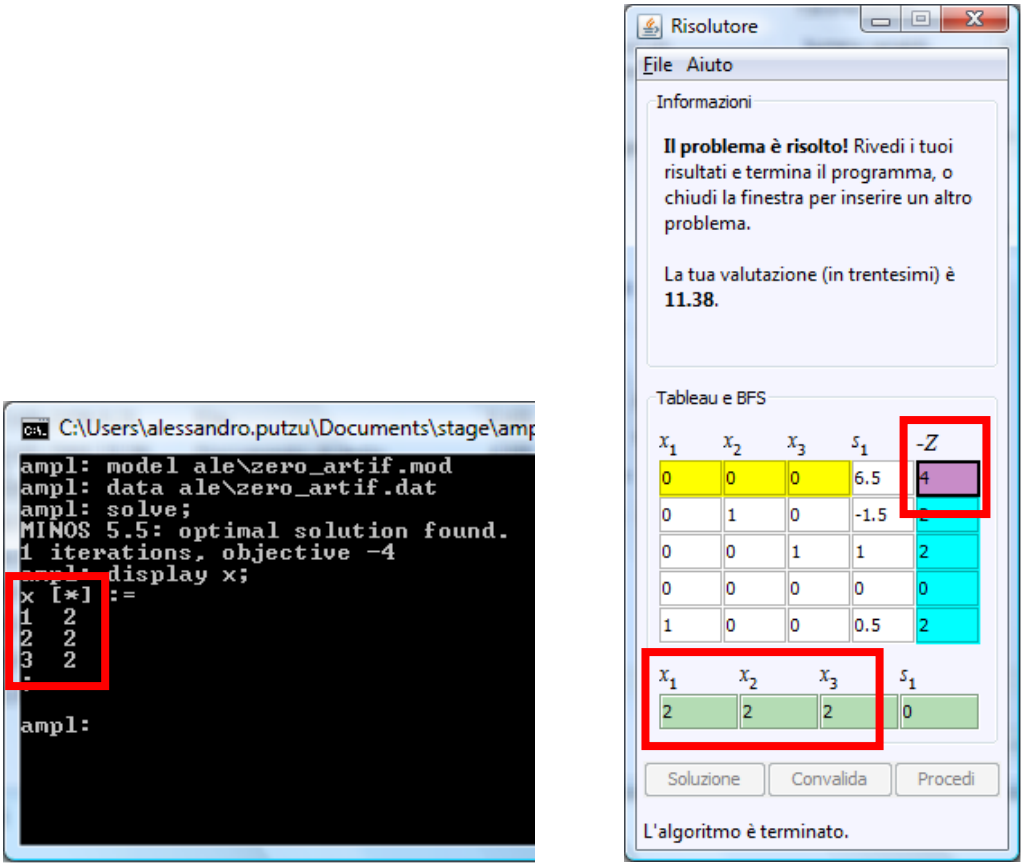


Fig. 38: Console di ampl.exe e schermata finale del risolutore per il problema di test n. 3

§5.5 – Caso di test n. 4: Problema illimitato e problema impossibile

Il quarto test consiste nella prova di due problemi, uno illimitato e l'altro inammissibile. Entrambi sono stati costruiti in modo che queste rispettive caratteristiche fossero chiare sin dalla formulazione algebrica.

Il primo è il seguente, con due variabili e un solo vincolo:

$$\text{Max. } Z = 2x_1 + x_2$$

$$\text{s.t.}$$

$$3x_1 \leq 6$$

$$x_{1\ldots2} \geq 0$$

E' immediato notare che, in un problema così formulato, la variabile x_2 non è soggetta ad alcun vincolo, se non quello di non negatività; ed essendo il problema di

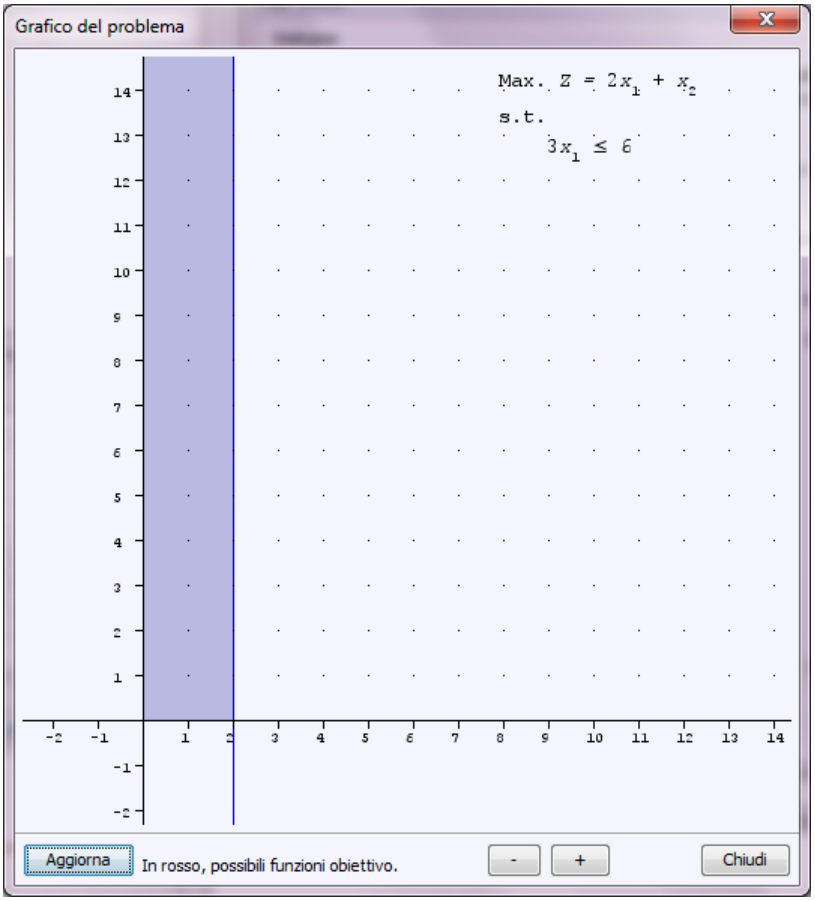


Fig. 39: Grafico del problema illimitato

massimizzazione, il valore dell'ottimo potrà crescere indefinitamente. Il grafico in **Fig. 39** conferma questa situazione: la regione ammissibile si estende indefinitamente verso l'alto.

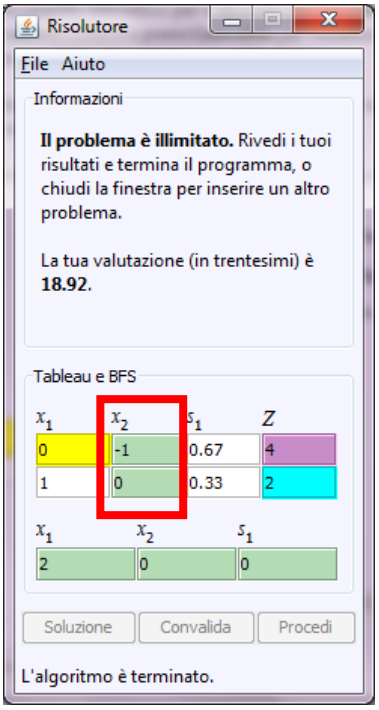


Fig. 40: Schermata finale per il problema illimitato

In **Fig. 40**, è possibile vedere come lo svolgimento del problema si interrompe con l'impossibilità di eseguire il **test del minimo rapporto**, evidenziata nella figura, che è condizione sufficiente perché un problema sia illimitato.

Il secondo problema è il seguente.

$$\text{Max. } Z = 2x_1 + x_2$$
$$\text{s.t.}$$
$$x_1 + 3x_2 \leq 6$$
$$x_1 + 3x_2 \geq 9$$
$$x_{1\ldots2} \geq 0$$

Come si può notare, i vincoli sono rette parallele e i semipiani da ciascuno di loro, non comprendono l'altro vincolo; la regione ammissibile è perciò vuota.

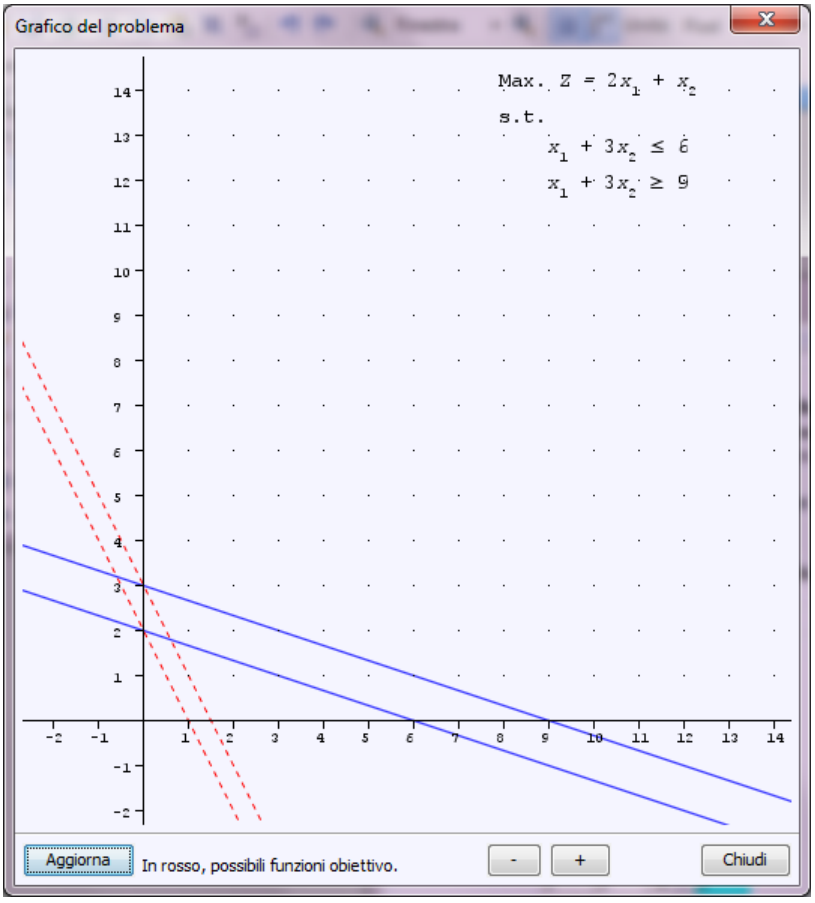


Fig. 41: Grafico del problema di test inammissibile

Anche in questo caso, il grafico in **Fig. 41** mostra la situazione di inammissibilità.

Fig. 42: Schermata finale del risolutore per il problema inammissibile

L'esecuzione si arresta alla fine della Fase Uno, quando il *tableau* ottimo presenta una variabile artificiale in base e diversa da 0, come mostrato in **Fig. 42**. La condizione di inammissibilità è stata evidenziata in figura.

Per questo caso di test, non è stato presentato l'*output* di **AMPL**, in quanto scopo della validazione era dare un riscontro visivo delle condizioni di inammissibilità o *boundlessness*.

§5.6 – *Caso di test n. 5: Problema di grandi dimensioni (stress test)*

L'ultimo test riguarda la risoluzione di un problema delle dimensioni massime consentite dal *tool*, 7 variabili e 7 vincoli, a coefficienti casualmente generati, di cui alcuni nulli e alcuni negativi. Tutti i vincoli sono di **maggioranza**, per generare un *tableau* il più grande possibile. Questo è il problema utilizzato per lo *stress test*:

$$\text{Min. } Z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$$

s.t.

$$4x_1 + 5x_2 + 7x_3 + 8x_4 + 6x_5 + 5x_6 + 78x_7 \geq 6$$

$$-5x_1 + 5x_2 - 6x_3 + 8x_4 - 9x_5 + 7x_7 \geq 4$$

$$5x_1 + 8x_2 + 9x_3 + 90x_4 + 8x_5 + 7x_6 + 5x_7 \geq 7$$

$$9x_1 - 7x_3 - 67x_4 + 5x_5 + 5x_6 + 45x_7 \geq 4$$

$$7x_1 - 6x_2 + 5x_3 - 5x_4 + 8x_5 - 9x_6 + 9x_7 \geq 5$$

$$-5x_1 + 45x_2 + 7x_3 - 9x_4 - 8x_6 + 6x_7 \geq 4$$

$$7 + 9x_2 + 0.7x_3 + 6x_4 - 4x_5 + 5x_6 + 6x_7 \geq 2$$

$$x_{1\ldots7} \geq 0$$

L'*output* di **ampl.exe** per questo problema, è rappresentato in **Fig. 43**.

Fig. 43: Console di ampl.exe per il problema di test n. 5

I valori delle variabili decisionali sono di ordine di grandezza molto piccolo, al limite dei valori visualizzabili dai campi in **Simply**.

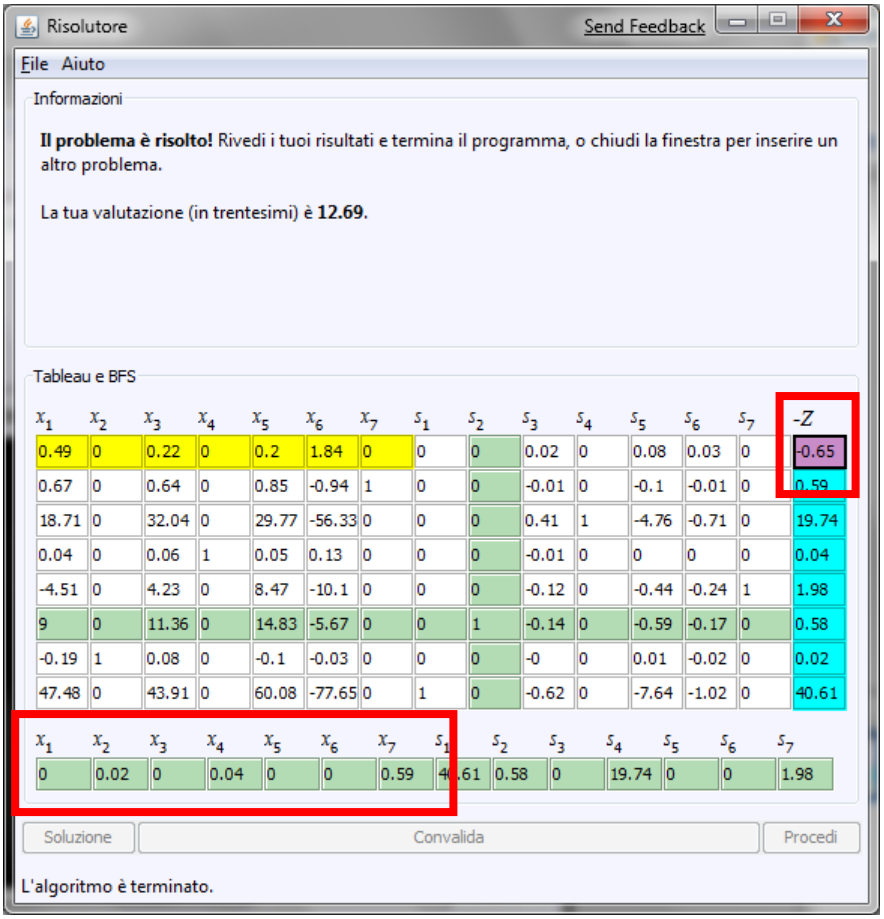


Fig. 44: Schermata finale del risolutore per il problema di test n. 5

Durante l'esecuzione di questo problema sul *tool*, è stato possibile notare dei valori "-0" (*meno zero*) nei campi del *tableau*. Non si tratta di un errore, ma di numeri negativi piccoli (minori di -0.005) rappresentati nei campi di testo, che arrotondano ogni valore contenuto in essi alla seconda cifra decimale; il valore del *set* interno (vedi **§4.4.2**) rimane comunque inalterato.

La schermata finale per questo problema è riportata in **Fig. 44**. I valori, seppure arrotondati, rispondono alle aspettative.

§5.7 – Considerazioni sui test eseguiti

I problemi inclusi nei vari casi di test, fin qui esaminati, hanno avuto lo scopo di dimostrare le funzionalità del *tool* **Simply**. Non affermano in alcun modo la correttezza in senso assoluto del programma.

Si tenga conto che quello presentato nelle pagine precedenti, è solo un campione dei problemi su cui è stato provato il *tool*. Ogni parte del programma, partendo dai metodi

di manipolazione della classe **Matrix** sino ad arrivare ai campi di testo dell'interfaccia grafica, è stata testata in maniera approfondita sin dal suo primo completamento e continuamente per tutto il restante percorso di sviluppo, con l'ausilio degli strumenti di *debug* forniti dall'ambiente di sviluppo **Eclipse**. In questo modo, ci si è potuti assicurare della funzionalità non solo delle singole parti, ma anche del progetto come un insieme.

Conclusioni

Nella presente tesi è stato sviluppato uno strumento per la risoluzione di problemi di Programmazione Lineare con il **metodo del simplesso**, con il preciso scopo di aiutare lo studente di Ricerca Operativa alla risoluzione di questi problemi.

Lo studio della letteratura ha portato alla considerazione che, sebbene esistano molti strumenti, per risolvere problemi di Programmazione Lineare, nessuno di essi raggiunge gli obiettivi prefissati: **semplicità d'uso**, **interattività**, **finalità didattiche**.

Per ognuno dei *tool* esaminati, sono state isolate le caratteristiche principali e migliori; ne sono state inoltre annotate le mancanze ed è stato studiato il modo per sopperire ad esse, nello strumento in costruzione. Si è poi proceduto alla strutturazione e implementazione di quest'ultimo, incorporando sia le caratteristiche più importanti dei *tool* esaminati, che la soluzione alle loro mancanze, sempre nel rispetto degli obiettivi prefissati.

Lo strumento sviluppato ha preso il nome di **Simply**. Può essere usato per risolvere problemi di **Programmazione Lineare Continua**, di massimizzazione o minimizzazione, a vincoli di maggioranza, minoranza o uguaglianza. Lo studente è limitato a 7 variabili e altrettanti vincoli. Questi sono gli obiettivi principali raggiunti:

- **Interattività**. Lo strumento definisce un percorso che lo studente deve seguire, ma è lo studente stesso a definire i tempi di esecuzione. Ogni passo implementato dell'**algoritmo del simplesso** richiede un'azione interattiva, che si presenta nella forma di risposta chiusa o inserimento di valori nel *tableau*.

- **Semplicità d'uso**. Si è cercato di mantenere l'interfaccia ordinata e facilmente utilizzabile, con una curva d'apprendimento minima. Numerosi, ma non eccessivi, messaggi di stato, indicano la situazione attuale del problema e danno informazioni minime sulle azioni da intraprendere per proseguire nello svolgimento del problema. L'interfaccia è **modale**: ogni stato è coerentemente

segnalato da messaggi, testuali e visivi, appropriati alla situazione, per minimizzare le possibilità di confusione.

- **Finalità didattiche**. Lo strumento ha, come si è detto, lo scopo di **guidare** lo studente nella risoluzione del problema; non quello di risolvere il problema al posto dello studente. A richiesta, e solo dopo almeno un tentativo per ogni fase che richieda l'inserimento di un *tableau*, è possibile ottenere la **soluzione**: questo, per evitare che lo studente non riesca più a procedere, il che diminuirebbe l'efficacia complessiva del *tool*. E' stata anche introdotto un meccanismo di **valutazione**, che offre, per ogni esercizio completato, un voto indicativo in trentesimi. Il voto ha valore di confronto, per fornire la misura del proprio miglioramento in successivi esercizi svolti.

- **Portabilità**. Il *tool* è stato sviluppato in linguaggio Java, compilato e distribuito come archivio **jar** auto-contenuto ed eseguibile su ogni piattaforma. L'unico requisito è un **JRE**, versione pari o superiore a **1.6.0_12**.

Sono inoltre state aggiunte funzionalità che aumentano l'utilità del programma in un ambito didattico: la possibilità di **esportare il problema** inserito in linguaggio **AMPL** e la possibilità, peraltro non inedita, di **visualizzare il grafico** per problemi fino a due variabili.

Il *tool* **Simply** è stato infine testato con varie tipologie di problemi, sia facilmente incontrabili durante una lezione o esercitazione di Ricerca Operativa, sia adatti a testare le funzionalità del programma al massimo livello possibile (ad es. un problema con 7 variabili e 7 vincoli). I test, effettuati confrontando i risultati del *tool* con quelli ottenuti caricando i *file* **AMPL** esportati, hanno mostrato la correttezza del programma nelle situazioni esaminate.

§6.1 – Possibili sviluppi del tool

Lo strumento sviluppato durante il lavoro di tesi, copre solo una parte della Programmazione Lineare. Sarà possibile, in futuro, aggiungere al programma delle estensioni che implementino ulteriori funzionalità, aumentandone così l'utilità

didattica.

Una di queste, è la possibilità di effettuare la **risoluzione del problema per via grafica**, per problemi di due variabili, spostando il valore della funzione obiettivo da un vertice all'altro fino al raggiungimento dell'ottimo.

Inoltre, si potrebbe aggiungere la possibilità di risolvere problemi di **Programmazione Lineare Intera**. Il *tool*, in questo modo, potrebbe essere usato con un raggio più ampio di esercizi. A questa classe di possibili aggiunte appartiene anche il calcolo del **duale**.

Infine, oltre alla possibilità di **esportare** problemi in linguaggio **AMPL**, l'aggiunta di ulteriori opzioni (ad es. il linguaggio **MPS**) aumenterebbe le opportunità di interfacciare il programma con altri, più famosi e sofisticati, che facciano uso di questi standard per la rappresentazione matematica. In maniera analoga, la possibilità di **salvare** e **caricare** problemi (in un formato standard, oppure specifico del programma) ne aumenterebbe la semplicità d'uso, per esempio in caso di esercizi particolari, in cui si voglia testare il proprio miglioramento in diverse sessioni di esercitazione.

Bibliografia

[Hillier & Lieberman, 2005] – Frederick S. Hillier, Gerald J. Lieberman, **Introduzione alla ricerca operativa**, VIII Edizione, McGraw-Hill, 2005

[Roma et al., 2008] – Roma, Lucidi, Facchinei, Palagi, **Appunti di Ricerca Operativa** per la facoltà di **Ingegneria Informatica**, Università La Sapienza di Roma, liberamente disponibili all'indirizzo http://www.dis.uniroma1.it/~or/RO/

[Cormen et al., 2001] – Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, **Introduzione agli Algoritmi, II Edizione**, McGraw-Hill, 2001.

[Archetti et al., 1989] – F. Archetti, E. Fagiuoli, A. Sciomachen, **Metodi della Ricerca Operativa**, G. Giappichelli Editore, 1989

[Spielman & Teng, 2001] – Daniel Spielman, Shang-Hua Teng, **Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time**, Annual ACM Symposium on Theory of Computing, New York 2001, liberamente disponibile all'indirizzo http://portal.acm.org/citation.cfm?doid=380752.380813

[Savitch, 2008] – Walter Savitch, **Absolute Java, Third Edition**, Pearson International, 2008

[McConnell, 2008] – Steve McConnell, **Code Complete – A Practical Handbook of Software Construction, Second Edition**, Microsoft Press, 2004

[Sun Inc., 2008] – Sun Microsystems, Inc., **Creating a GUI with JFC/Swing**, disponibile all'indirizzo http://java.sun.com/docs/books/tutorial/uiswing/, ai termini d'uso contenuti nella pagina *web* http://www.sun.com/termsofuse.jsp#g2_12

Multas per gentes et multa per aequora vectus, *eccomi giunto alla fine di questo percorso. Molto è stato l'entusiasmo iniziale, la sfiducia incontro alle difficoltà, la corsa finale e lo slancio. Impossibile ricordare gli anni passati tra questi edifici, senza pensare alle persone e alle cose che mi hanno aiutato e sostenuto, ognuno a suo modo, durante il cammino.*

Ringrazio per primi i miei genitori, che mi hanno dato la possibilità di accedere e frequentare l'università, senza dovermi mantenere da solo gli studi. Dimostrazione che non serve "andai a sa scola" *per essere persone gentili.*

Ringrazio Federica, che è entrata nella mia vita al momento giusto per darmi quella spintarella necessaria a riprendere le redini, allora molto allentate, della mia carriera universitaria, e non ne è più uscita. Auguri a noi! E grazie per il bellissimo logo di Simply!

Inoltre, naturalmente, grazie a Ilaria e alla prof.ssa Messina, per avermi dedicato il loro tempo prezioso. E anche a Maria Grazia Vanola, chiave di volta del Dipartimento!

Un grazie anche a tutti gli amici, quelli che c'erano già da prima e quelli che sono arrivati poi, quelli che sono rimasti e quelli che ho perso per strada. Loro sono le uscite in piazza, i pomeriggi sui libri e i progetti che finivano esattamente al momento giusto, le lavagne piene di scritte, le foto al cellulare con gli emoticon in live action, *i ritornelli e nutella (o nastrine e nutella, a seconda… o branzino e nutella, perché no), la cavolata delle tre di notte, l'insalata e le pecore, l'alba delle cinque del mattino, la ballerina che gira, le discussioni interminabili su giochi, hardware e software, l'aeroporto e la briscola chiamata, le partite a scopa d'assi al circolino (tanto tempo fa) e tutto il resto che forse ora non ricordo, ma che è e sarà sempre. Un miscuglio di anni e persone, in ordine sparso, e così dev'essere. E ovviamente, non mancano i compagni tesisti (e non) con cui ho diviso il laboratorio della chiave col fiocco rosso, in questi ultimi mesi!*

Non poteva mancare un grazie alle grandi personalità del mondo dell'informatica che hanno, chi più chi meno, guidato il mio pensiero. Il maestro Edsger Dijkstra, delle cui idee farò sempre buon uso, e coloro che hanno lasciato pagine tanto piene di sapienza, da brillare di luce propria: Cormen, Leiserson e Rivest, McConnell, Stroustrup, Kernigham & Ritchie, Savitch…

E grazie, infine, alle cose che sono parte di me non meno della materia del mio corpo. La mia camera, con infiniti ninnoli e aggeggi, alcuni nuovi e alcuni vecchi, ma ognuno egualmente caro; il mio computer, che ultimamente ho cambiato, ma che mi ha fedelmente servito per lunghi innumerevoli anni, i libri del grande Tolkien, i miei quaderni, la fantascienza, Dalì, le bic nere, Lupo Ernesto e Faccia Intelligente, i giochi.