PROGETTI GNCS 2005:
RICHIESTA DI FINANZIAMENTO

1 Titolo del progetto

Sviluppo di risolutori di vincoli e loro applicazioni.

2 Coordinatore

Agostino Dovier (Udine, mesi 5)

3 Elenco dei partecipanti e sede di appartenenza

Strutturati:

1. Chiaruttini Claudio (Trieste, mesi 4)
2. Fabris Francesco (Trieste, mesi 4)
3. Fogolari federico (Udine, mesi 4)
4. Formisano Andrea (L'Aquila, mesi 4)
5. Franceschet Massimo (Pescara, mesi 4)
6. Luccio Flaminia (Trieste, mesi 4)
7. Montanari Angelo (Udine, mesi 4)
8. Omodeo Eugenio (Trieste, mesi 4)
9. Piazza Carla (Udine, mesi 4)
10. Rossi Gianfranco (Parma, mesi 4)
11. Sgarro Andrea (Trieste, mesi 4)

Non strutturati:

12. Bortolussi Luca (Udine, mesi 6)
13. Bresolin Davide (Udine, mesi 6)
14. Casagrande Alberto (Udine, mesi 6)
15. Dal Fabbro Cristian (Udine, mesi 3)
16. Dal Palù Alessandro (Udine, mesi 6)
17. Gentilini Raffaella (Udine, mesi 6)
18. Gubiani Donatella (Pescara, mesi 6)
19. Puppis Gabriele (Udine, mesi 6)
20. Scalabrin Simone (Udine, mesi 6)
21. Vitacolonna Nicola (Udine, mesi 6)
22. Zantoni Marco (Udine, mesi 6)

4 Il Progetto

4a–La base di partenza

La Programmazione con Vincoli è una metodologia di programmazione che permette di formalizzare il problema da risolvere in modo puramente dichiarativo con l'ausilio di formula logiche (dette *vincoli*). La fase di computazione è tipicamente basata sulla ricerca di una o più soluzioni che soddisfino l'insieme di vincoli imposti. Per velocizzare l'esecuzione la filosofia è quella di inserire in modo incrementale, senza snaturare il codice dichiarativo, altri vincoli che permettano di indirizzare la ricerca. La programmazione con vincoli è diventata recentemente praticabile mediante la commercializzazione di alcuni linguaggi. In particolare i linguaggi logici SICStus Prolog ed ECLiPSe permettono di programmare in modo dichiarativo utilizzando la filosofia *constraint + generate* [17, 1], nata in contrapposizione all'inefficiente metodologia di programmazione dichiarativa *generate & test*. Tali linguaggi, detti linguaggi *CLP* (da Constraint Logic Programming [16]) mettono a disposizione svariati domini e risolutori di vincoli e sono stati impiegati con successo nella risoluzione di problemi di ottimizzazione o di ricerca di soluzioni ammissibili per problemi in domini complessi e con funzioni obiettivo arbitrarie.

Più in dettaglio, un vincolo è tipicamente una formula logica in un dato alfabeto Σ. Tale formula deve essere interpretata in un dato dominio. Ad esempio, la formula

$$1 < X \wedge X < Y \wedge Y < 2$$

è un vincolo soddisfacibile interpretando le variabili ed il simbolo di predicato $<$ in $\mathbb{Q}$ o in $\mathbb{R}$ ma non in $\mathbb{N}$. Un linguaggio di programmazione con vincoli mette a disposizione del programmatore varie classi di vincoli e di domini su cui interepretarli, nonchè, per ogni dominio concesso, un *risolutore di vincoli* che permette di semplificare le formule stesse e di riconoscere la loro eventuale insoddisfacibilità. Inoltre, per ogni classe di vincoli vi sono tipicamente alcune funzioni built-in quali ad esempio funzioni di minimizzazione o di ricerca di soluzioni per le variabili in gioco. Ad esempio, il vincolo in $\mathbb{N}$:

$$X \in [2,5] \wedge Y \in [1,4] \wedge X < Y$$

è soddisfacibile, e può essere facilmente semplificato (usando tecniche standard di propagazione di vincoli) nel vincolo equivalente:

$$X \in [2,3] \wedge Y \in [3,4] \wedge X < Y$$

Il vincolo racchiude in sé tutte le sue soluzioni in modo implicito. Se siamo interessati ad esplicitare una soluzione, un modo è quello di richiedere la ricerca delle soluzioni con il predicato `labeling`. Ad esempio:

$$X \in [2,5] \wedge Y \in [1,4] \wedge X < Y \wedge \texttt{labeling}([X,Y])$$

restituisce (in modo non-deterministico) i valori:

$$X = 2, Y = 3 \qquad X = 2, Y = 4 \qquad X = 3, Y = 4$$

Se invece volessimo conoscere la soluzione che minimizza una data funzione (ad esempio $2X^3 - Y^2$—si osservi che non si tratta di una funzione lineare) dovremmo fornire un *goal* del tipo:

$$X \in [2,5] \wedge Y \in [1,4] \wedge X < Y \wedge \texttt{labeling}([\texttt{minimize}(2 * X^3 - Y^2)], [X,Y])$$

ottenendo in output:

$$X = 2, Y = 4$$

Varie classi di domini, vincoli e di tecniche di risoluzione degli stessi sono state sviluppate per la programmazione con vincoli. Per molte altre interessanti classi invece non sono ancora disponibili degli adeguati risolutori. Inoltre, la chiusura del codice per le classi di vincoli inglobate nei linguaggi commerciali non permette all'utente, una volta sviluppato un prototipo funzionante per una specifica applicazione, di specializzarlo all'applicazione stessa, ottimizzandone le prestazioni. In particolare, sarebbe necessario poter accedere alla struttura dati ove i vincoli vengono memorizzati a basso livello. Il progetto vuole inserirsi in questo contesto: si vuole sia sviluppare dei risolutori per classi non ancora disponibili che migliorare, guidati da applicazioni specifiche, risolutori di classi esistenti.

Nel seguito si cercherà di entrare un pò più in dettaglio su alcune classi di vincoli rispetto alle quali emergono le competenze dei vari proponenti il progetto.

Vincoli su domini finiti. L'utilizzo di vincoli su domini finiti (in breve, $CLP(\mathcal{FD})$), quali ad esempio quelli dell'esempio riportato sopra, permette di formulare problemi di ottimizzazione combinatoria, quali problemi di scheduling, timetabling, etc. Si veda ad esempio [17, 1] per una dettagliata introduzione ed analisi a questa metodologia di programmazione. I risolutori di vincoli per tali domini sono basati:

- sul concetto di propagazione di vincoli, utilizzando tipicamente la tecnica di *arc consistency* o sue semplificazioni, e
- sulla ricerca nello spazio delle soluzioni, controllata dalla propagazione dei vincoli che permette di rilevare assegnamenti parziali delle variabili che non possono condurre a soluzioni accettabili.
- Per problemi di ottimizzazione, la ben nota tecnica di *branch-and-bound* collabora con la propagazione di vincoli per aumentare il numero di rami tagliati.
- Per problemi in cui una soluzione approssimata è sufficiente (ad esempio problemi reali di allocazione risorse ove la soluzione teoricamente ottima potrebbe essere inficiata dal guasto di una macchina utensile o da un'assenza per malattia) sono presenti tecniche approssimate per la scansione dell'albero di ricerca, quali ad esempio la *limited discrepancy search.*

Recentemente in [2] è mostrato come utilizzare in modo naturale la programmazione logica con vincoli su domini finiti per un problema reale di allocazione di sorgenti acustiche su territorio urbano. In [4] è illustrato come tale metodologia di programmazione permetta di codificare, ottenendo risultati promettendi, il problema del protein folding. In questo caso tuttavia emergono problemi per lo scarsa visibilità della struttura dati "vincolo" concessa all'utente. Per proteine di lunghezza intorno al centinaio risulta indispensabile inserire euristiche nella ricerca di soluzioni legate strettamente al problema, ma ciò è impossibile senza l'accesso diretto alla costruzione dell'albero di ricerca.

Vincoli su intervalli. Affini ai vincoli su domini finiti sono i vincoli su intervalli. Il fatto che una variabile X possa assumere valori nell'intervallo $[a, b]$ è un tipico vincolo da domini finiti. Che il suo valore però stia nell'unione o intersezione tra più intervalli o che tali intervalli possano aumentare durante la computazione esula di solito dalle capacità dei risolutori su domini finiti. Inoltre potremmo essere interessati ad intervalli di $\mathbb{R}$ anziché di $\mathbb{N}$. La programmazione logica con vincoli su intervalli ha diverse naturali applicazioni, ad esempio nel riconoscimento di immagini, dove i punti identificati da sensori e telecamere vengono forniti non in modo preciso, ma come elementi di intervalli reali. I risolutori attualmente disponibili non permettono di minimizzare/massimizzare funzioni arbitrarie con variabili definite su intervalli. Il problema è sicuramente molto complesso, ma una sua soluzione approssimata mediante ripetuta bisezione degli intervalli appare possibile.

.... Gli intervalli sono naturali nel ragionamento temporale, dove la decidibilità di varie logiche è stata studiata [x Angelo] .....

Vincoli su insiemi. Sono state studiate le possibilità concesse dall'utilizzo di vincoli di tipo insiemistico quali ad esempio

$$X \in \{a\} \cup Y \wedge Y \subseteq \{a, b\} \wedge X \notin \{b\}$$

Linguaggi logici con vincoli di tipo insiemistico permettono un elevato livello di astrazione in cui sviluppare prototipi funzionanti per risolvere un dato problema. Ad esempio, il problema di stabilire se un grafo i cui nodi sono $\{X_1, \ldots, X_n\}$ e archi $\{X_{i_1}, X_{j_1}\}, \ldots, \{X_{i_k}, X_{j_k}\}$ possa o meno essere colorato mediante 3 colori a, b e c in modo tale che nessun nodo sia adiacente ad un nodo dello stesso colore può essere espresso mediante un unico vincolo:

$$\{\{X_{i_1}, X_{j_1}\}, \ldots, \{X_{i_k}, X_{j_k}\}\} = \{\{a, b\}, \{a, c\}, \{b, c\}\}$$

Lo studio di vincoli insiemistici e delle relative problematiche è una branca della Teoria Computabile degli Insiemi [3]. In tale contesto è stato presentato e sviluppato il linguaggio logico con vincoli $CLP(\mathcal{SET})$ [9, 11]. Diversi risultati nell'area dei risolutori di vincoli per insiemi/iperinsiemi sono stati ottenuti dai partecipanti al progetto. In [13] sono definiti in modo parametrico algoritmi di unificazione per termini che rappresentano insiemi/multi-insiemi e liste compatte. In [8] viene presentato un algoritmo di unificazione per iperinsiemi e in [7] per insiemi *alla Quine* con ur-elementi. In [5] il linguaggio $CLP(\mathcal{SET})$ viene combinato con il linguaggio $CLP(\mathcal{FD})$ ottenendo un linguaggio con vincoli su più domini che permetta di trarre vantaggio dalla dichiaratività dei vincoli insiemistici e dall'efficienza dei risolutori di vincoli su domini finiti. In [10] sono studiati e risolti i vincoli di teorie insiemistiche basate su un simbolo di funzione ACI. In [14] sono studiate le problematiche relative all'introduzione della negazione costruttiva in linguaggi con vincoli su domini insiemistici.

Vincoli per il DNA word design. Un settore in cui la programmazione con vincoli può dare interessanti contributi è la cosiddetta *DNA word design*. Il problema è quello di identificare un insieme di frammenti di DNA di lunghezza fissata che possiedano caratteristiche tali da renderli adatti per la computazione molecolare. Questo è un nuovo e promettente settore della scienza, al confine tra biologia ed informatica, che cerca di sfruttare l'intrinseco parallelismo e altre proprietà delle molecole di DNA per effettuare delle computazioni. In particolare, queste tecniche sono basate sull'ibridazione (cioè l'unione di frammenti complementari) di particolari filamenti di DNA, che codificano i dati del problema. Da qui la necessità di utilizzare frammenti di DNA che ibridizzano sempre nel modo voluto e non creino configurazioni spurie. In altri termini, c'è la necessità di identificare sottoinsiemi di stringhe di DNA di lunghezza fissata, che soddisfino particolari vincoli, di solito espressi con misure che sono variazioni sul tema della distanza di Hamming. Finora, tutti gli approcci costruttivi in questa

direzione hanno utilizzato algoritmi di ricerca stocastica. In [20] è stata sviluppata una cornice teorico-informazionale, basata sul concetto di distinguibilità, nella quale hanno riscritto il problema di trovare codici ottimi di parole di DNA, suggerendo che il processo di ibridazione può essere visto come una sorta di "comunicazione".

Questo approccio normativo può essere esteso e completato da algoritmi che costruiscono direttamente questi codici di DNA. Un contesto naturale per realizzare questo obiettivo è la programmazione con vincoli, dove non solo si possono modellizzare facilmente tali problemi, ma anche la distinguibilità stessa può diventare un vincolo, con delle proprie procedure di risoluzione.

Formalmente, dato un insieme $\mathcal{X}^n$ di stringhe di lunghezza n nell'alfabeto $\{a, c, g, t\}$ e una misura di diversità (distanza) $d : \mathcal{X} \times \mathcal{X} \to \mathbb{N}$, la distinguibilità tra due parole di codice $a, b \in \mathcal{X}^n$ è definita come $\delta(a, b) = \min_{c \in \mathcal{X}^n} \max\{d(a, c), d(b, c)\}$, cioè mediante un problema di ottimizzazione. Dato un codice $\mathbb{C} \subset \mathcal{X}^n$, la sua distinguibilità minima è $d_{min}(\mathbb{C}) = \min_{c,c' \in \mathbb{C}} \delta(c, c')$, e il vincolo per un codice affidabile è $d_{min}(\mathbb{C}) \geqslant \tau$, per una qualche soglia τ. Infine bisogna massimizzare la cardinalità di $\mathbb{C}$. Stante il fatto che la distinguibilità congiunta di due o più distanze può essere facilmente definita, è chiaro che sfruttando questa formalizzazione, tutti i diversi vincoli del problema originale possono essere sussunti in un vincolo solo, che coinvolge la distinguibilità.

Con un buon risolutore di vincoli, sviluppato "ad hoc" per questa quantità, c'è la possibilità di sviluppare un efficiente algoritmo per costruire dei codici ottimi.

Parallelamente, un'altra strada da seguire può essere quella di dimenticarsi delle distinguibilità e di modellare i vincoli del problema direttamente in termini delle distanze originali (come la misura H e altre, si veda [18] per una rassegna). Questo dà origine ad un problema diverso, che può essere comunque risolto all'interno della cornice della programmazione con vincoli.

Le possibilità di utilizzo della programmazione con vincoli (in particolare di tipo multiinsiemistico) per il DNA computing sono studiate in [12].

Identificazione di segnali biologici nel genoma. Un problema molto difficile in bioinformatica è l'identificazione di segnali biologici nel genoma (si veda [19] per una panoramica). In particolare, c'è la necessità di identificare piccoli frammenti di DNA, chiamati TFBS, che sono situati nelle regioni non codificanti di un gene e che costituiscono dei siti attivi nelle attività di regolazione. Generalmente si parte da un insieme di geni corregolati, o supposti tali, e si vogliono identificare i TFBS responsabili di questa attività comune. Queste regioni sono lunghe da 8 a 20 nucleotidi, e mostrano delle differenze tra di loro, cioè sono ad una distanza di Hamming o di Levenshtejn positiva. Generalmente, la loro distanza relativa da una sequenza consenso comune è dell'ordine del 25–30%. Se si ricercano direttamente le occorrenze di questa stringa ignota, si devono cercare sequenze a distanza relativa reciproca dell'ordine del 50–60%, trovando così un enorme quantità di soluzioni spurie. È quindi necessario sempre un ulteriore filtraggio, atto a discriminare tra soluzioni buone e soluzioni

cattive, e basato su una qualche misura statistica.

Per mezzo della programmazione con vincoli, il nostro obiettivo è di integrare l'attività di ricerca e la successiva elaborazione in un'unica fase. Il dominio della nostra ricerca può essere formalizzato nel seguente modo: siano $s_1, \ldots, s_k$ delle stringhe e sia R l'insieme delle loro sottostringhe (r, i), dove con questa notazione si intende che r è una sottostringa di s_i. Dunque il dominio è $\mathcal{X} = 2^R$, l'insieme potenza di R. Su $\mathcal{X}$ si devono imporre diversi vincoli, relativi alla lunghezza delle sottostringhe ricercate, al loro numero di occorrenze, al numero di stringhe in cui occorrono, alla distanza relativa. Inoltre, si possono anche imporre dei vincoli che derivano dalle misure statistiche usate comunemente, realizzando così la citata integrazione tra fase di ricerca e fase di discriminazione.

Tutti questi vincoli devono essere propagati in modo tale da ridurre drasticamente lo spazio di ricerca, e conseguentemente i propagatori di vincoli devono essere sviluppati "de novo".

Con questo approccio si mira a sviluppare degli algoritmi competitivi per la ricerca di TFBS, in quanto i vincoli derivanti da indici statistici non sono mai stati usati, finora, per velocizzare il processo di identificazione di insiemi di sottostringhe candidate ad essere degli elementi regolatori.

Relazioni tra Constraint Programming e ASP. Nell'area della programmazione dichiarativa sta riscuotendo un crescente in- teresse il cosiddetto *Answer Set Programming* (in breve, ASP). I programmi ASP sono programmi costituiti da clausole e goals con arbitraria presenza di letterali negati. Non possono invece essere utilizzati simboli di funzione non costante. Ciò garantisce finitezza del Dominio di Herbrand e della Base di Herbrand dove cercare gli eventuali modelli del programma stesso. La forma dei programmi fa sí che non sia garantita l'esistenza di un modello ed in particolare di un unico modello minimo. I modelli considerati "buoni" sono i modelli stabili [15]. Il calcolo degli stessi avviene bottom-up mediante opportuni risolutori (quali ad esempio SMODELS, DLV, ASSAT). Mediante ASP è naturale programmare problemi di planning che per loro natura contengono parecchie negazioni, che mal si adatterebbero alla risoluzione tradizionale top-down permessa dalla SLD-risoluzione dello standard logic programming. Tuttavia è dimostrato che la classe dei problemi codificabili mediante ASP equivale esattamente alla classe NP [6]. Pertanto nei risolutori di ASP vi sono tecniche di risoluzione simili a quelle utilizzate, ad esempio, nei risolutori di vincoli $CLP(\mathcal{FD})$ che tipicamente affrontano problemi simili. Uno degli obiettivi del progetto è di comparare le due famiglie di risolutori, e sperabilmente sviluppare un ambiente unificante che prenda il meglio delle due classi.

4b–Le metodologie e gli Obiettivi

Come introdotto nella sezione precedente, varie sono le tematiche da affrontarsi nel progetto. Una volta codificato (qualora non sia ancora stato fatto) un problema nell'adeguato paradigma di programmazione con vincoli, si cercherà di

ottimizzare il codice specializzando il risolutore al problema in esame (o meglio ad una famiglia di problemi ad esso collegati).

In particolare, nell'area dei *vincoli su domini finiti* il principale obiettivo è quello dello sviluppo di un risolutore di vincoli ad-hoc per vincoli su reticoli adatti a discretizzare le proteine, al fine di ottimizzare il codice presentato in [4]. [Dal Palu'–Dovier]

Nell'area dei Vincoli su intervalli, si intende sviluppare un risolutore di vincoli in grado di minimizzare (eventualmente in modo approssimato) funzioni reali con variabili vincolate a stare in intervalli. I vincoli non sono necessariamente di tipo lineare. [Bortolussi-Dovier] Si intende inoltre [x Angelo ...]

Nell'area dei vincoli su insiemi (e affini) si intende procedere ad investigare le problematiche connesse a linguaggi di programmazione con vincoli di tipo insiemistico/iperinsiemistico/multiinsiemistico. [Dovier–Formisano–Omodeo–Rossi]

Nell'area dei vincoli per il DNA word design, come spiegato nella sezione 4a, si intende modellare il problema in programmazione con vincoli ed eventualmente sviluppare un risolutore ad-hoc. [Bortolussi–Fabris]

Nell'area dell'Identificazione di segnali biologici nel genoma, come spiegato nella sezione 4a, si intende usare la programmazione con vincoli per integrare la fase di ricerca di segnali e la successiva fase di filtraggio. Anche in questo caso, sarà necessario sviluppare nuovi risolutori di vincoli. [Bortolussi–Sgarro]

Infine, si desidera integrare le idee utilizzate nei risolutori per answer set programming e nei risolutori per vincoli su domini finiti in $CLP(\mathcal{F}D)$ in un unico risolutore adatto ad affrontare, in generale, problemi NP-completi. [Dovier–Formisano]

4c–Gli aspetti innovativi

La programmazione con vincoli è di per sé una metodologia di programmazione innovativa. Lo sviluppo di risolutori di vincoli in nuovi domini permette di aprire questa metodologia a nuove applicazioni, in particolare, ma non escusivamente, di tipo multimediale e bioinformatico.

4c—Le attività previste

Per ottenere ciascuno dei principali obiettivi di cui alla sezione 4b sarà necessario una fase di acquisizione della bibliografia più recente da parte dei ricercatori coinvolti. Per ciascuno dei punti in questione sarà necessario dapprima classificare il problema da risolvere dal punto di vista della complessità computazionale. Si tratterà dunque di scegliere le strutture dati più opportune per la codifica efficente dei risolutori desiderati. Infine i codici ottenuti saranno integrati nei linguaggi con vincoli SICStus e ECLiPSe.

5 Richiesta Finanziaria

Ritenendo escluse, come sempre, le spese per materiale inventariabile richiediamo un contributo di 20.000 euro che, vista la numerosità del gruppo, il suo impegno in termini di mesi uomo ed il lavoro richiesto su vari fronti, sembra adeguato all'ampia varietà di iniziative necessarie a sostegno del progetto. Tali fondi saranno interamente utilizzati per coprire spese di missione.

References

[1] K. R. Apt. *Principles of Constraint Programming.* Cambridge, 2003.

[2] F. Avanzini, D. Rocchesso, A. Belussi, A. Dal Palù, and A. Dovier. Designing an urban-scale auditory alert system. *IEEE Computer*, 37(6):55–61, September 2004.

[3] D. Cantone, E. Omodeo, and A. Policriti. *Set Theory for Computing: From Decision Procedures to Declarative Programming with Sets.* Springer, 2001.

[4] A. Dal Palù, A. Dovier, and F. Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5(186):1–12, November 2004.

[5] A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. Integrating Finite Domain Constraints and CLP with Sets. In D. Miller, editor, *Proc. of Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 219–229. ACM Press, August 2003.

[6] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[7] A. Dovier, A. Formisano, and E. Omodeo. Decidability results for sets with atoms. *ACM Transaction on Computational Logic*, 2005. To appear.

[8] A. Dovier, E. Omodeo, and A. Policriti. Solvable set/hyperset contexts: Ii. a goal-driven unification algorithm for the blendedcase. *Applicable Algebra in Engineering, Communication and Computing*, 9(4):1–48, 1999.

[9] A. Dovier, E. G. Omodeo, E. Pontelli, and G. Rossi. {log}: A Language for Programming in Logic with Finite Sets. *Journal of Logic Programming*, 28(1):1–44, 1996.

[10] A. Dovier, C. Piazza, and E. Pontelli. Disunification in aci1 theories. *Constraints (International Journal)*, 9(1):35–91, 2004.

[11] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. *ACM Transactions on Programming Languages and Systems*, 22(5):861–931, 2000.

[12] A. Dovier, C. Piazza, and G. Rossi. Multiset constraints and p systems. In C. Calude, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View [Workshop on Multiset Processing, WMP 2000, Curtea de Arges, Romania, August 21-25, 2000]*, volume 2235 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2001.

[13] A. Dovier, A. Policriti, and G. Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae*, 36(2/3):201–234, 1998.

[14] A. Dovier, E. Pontelli, and G. Rossi. Constructive negation and constraint logic programming with set. *New Generation Computing*, 19(3):209–255, 2001.

[15] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[16] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[17] K. Marriott and P. J. Stuckey. *Programming with Constraints*. The MIT Press, 1998.

[18] G. Mauri and C. Ferretti. Word design for molecular computing: A survey. In *DNA Computing, 9th International Workshop on DNA Based Computers*, pages 37–46, 2003.

[19] G. Pavesi, G. Mauri, and G. Pesole. In silico representation and discovery of transcription factor binding sites. *Briefings in Bioinformatics*, 5(3):1–20, 2004.

[20] A. Sgarro and L. Bortolussi. Codeword distinguishability in minimum diversity decoding. In preparazione.