UNIVERSITA' DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

DIPARTIMENTO DI INGEGNERIA CIVILE EDILE AMBIENTALE

CORSO DI LAUREA IN INGEGNERIA MECCANICA

Tesi di Laurea Magistrale in Ingegneria Meccanica

MODELLAZIONE GEOMETRICA GENERATIVA ED OTTIMIZZAZIONE DI STRUTTURE CELLULARI PER LA FABBRICAZIONE MEDIANTE ADDITIVE MANUFACTURING

Relatore: Ch.mo Prof. Gianmaria Concheri

Correlatore: Ing. Gianpaolo Savio

Laureando: Flavio Gaggi

ANNO ACCADEMICO 2013/2014

Indice

1. Introduzione.

In molti campi l'esigenza di realizzare oggetti che presentino strutture alleggerite, ma al contempo che garantiscano adeguata rigidezza, spinge verso lo studio di strutture nella quale il materiale massivo sia sostituito da una struttura reticolare interna. Per esempio, nell'ambito biomedico, c'è assoluta necessità di realizzare parti alleggerite e al contempo altamente customizzate, come nella realizzazione di innesti ossei e protesi dentali. Per prima cosa occorre individuare una tecnologia in grado di produrre oggetti a superficie *free-form*, per riprodurre la forma delle parti del corpo da sostituire, che impieghi materiali biocompatibili e che permetta di realizzare strutture alleggerite internamente e sicuramente le tecniche *Additive Manufacturing*, AM, sono attualmente le più adatte per soddisfare le richieste. In particolare la tecnologia *Selective Laser Sintering*, SLS, basata sulla sinterizzazione laser di granuli finissimi del materiale base appare promettente.

Da quando le tecniche AM si sono sviluppate è stata posta grande attenzione nel perfezionamento della tecnologia di produzione, ma non altrettanta è stata posta sulle politiche di impiego del meteriale. L'importanza di quest'ulimo aspetto risiede nel fatto che, il volume di materiale impiegato, oltre ad essere uno dei principali driver di costo nella produzione, consente anche di controllare la rigidezza ottenuta e il peso dell'oggetto: si pensi ai benefici, in termini di bilanciamento dei pesi e della rigidezza, per il paziente a cui sia stata applicata una protesi del femore alleggerita rispetto ad una massiva. Data l'importanza di che potrebbe rivestire nelle applicazioni, la parte centrale del lavoro svolto ha riguardato lo sviluppo di procedure per la progettazione e ottimizzazione, in termini di volume di materiale impiegato, di strutture reticolari che si sostituiscano al materiale massivo.

Nel capitolo 2 si è affrontato lo studio dei cosiddetti materiali cellulari, ovverosia dei materiali non massivi costituiti dalla ripetizione regolare di celle elementari aperte che formano una stuttura reticolare di riempimento del volume interno dell'oggetto. In quest'ambito si sono definiti i concetti di mesostruttura, cioè la struttura reticolare di riempimento formata dalla ripetizione regolare di celle elementari ed esostruttura, il

guscio esterno della struttura. La ricerca ha riguardato le celle aperte in quanto, pensando alla fabbricazione con tecnica SLS, sono quelle che permettono l'eliminazione dal volume sotteso del materiale granulare non sinterizzato. Sempre nel capitolo 2 è presentata la descrizione della tecnica SLS e la spiegazione di come il *delay time* e i parametri tecnologici del processo possano influenzare la qualità del manufatto ottenuto con tale tecnologia.

Il capitolo 3 tratta l'introduzione agli algoritmi genetici e la descrizione del risolutore genetico Galapagos implementato nell'ambiente di lavoro Grasshopper, plug in del software di modellazione Rhinoceros5 per il disegno parametrico. Il risolutore genetico è impiegato nella procedura di ottimizzazione delle beam mediante algoritmo genetico per il calcolo della mesostruttura presentato nel capitolo 4.

Il capitolo 4 descrive le procedure sviluppate per il progetto e l'ottimizzazione di strutture cellulari. Le procedure sono sviluppate in ambiente Grasshopper e forniscono modelli virtuali in ambiente Rhinoceros5 in maniera automatizzata. La procedura iterativa di ottimizzazione delle *beam* e la procedura basata sull'algoritmo genetico per l'ottimizzazione delle *beam* definiscono la mesostruttura, impostando il tipo di cella e le dimensioni caratteristiche di cella e la ottimizzano, definendo i diametri degli elementi con l'obbiettivo di ottenere una struttura nella quale i livelli di utilizzazione degli elementi che la compongono siano per quanto possibile prossimi all'utilizzazione obbiettivo, solitamente impostata pari al 50%. La definizione di utitilizzazione dell'elemento è fornita nel paragrafo 4.1.5. I due approcci differiscono sostanzialmente nel fatto che il primo utilizza una procedura iterativa di tipo deterministico, mentre quello che impiega l'algoritmo genetico pe l'ottimizzazzione segue un l'approccio di tipo stocastico.

Oltre alle procedure di definizione della mesostruttura è presentata anche una procedura iterativa di ottimizzazione delle beam e delle shell, chiamata procedura B&S.

Nel capitolo 5 sono presentati dei *test case* per saggiare le capacità delle procedure sviluppate ed esposti i risultati ottenuti

2. Materiali cellulari.

Si dicono materiali cellulari quei materiali nei quali si può riconoscere una ripetizione degli elementi strutturali. Tra questi si annoverano gli schiumati, che hanno una struttura randomica oppure quelli caratterizzati da celle di forma e dimensioni ben definite, come ad esempio gli *honeycomb*. Questi materiali, prodotti per l'impiego in particolari settori quali l'isolamento, il filtraggio e la creazione di rinforzi leggeri e le loro controparti naturali, quali il legno, l'osso e le spugne, hanno strutture che permettono di ottimizzarne la resa in specifici ambiti. Tali strutture sono caratterizzate da elevate prestazioni specifiche in termini di rapporto tra proprietà e peso e per tale ragione sono molto studiate. Le utili proprietà che contraddistinguono i materiali cellulari dipendono dal materiale di cui sono costituiti, dalla densità relativa e dalla struttura geometrica che li caratterizza. E' fondamentale collegare le proprietà fisiche dei solidi cellulari alla densità e alla microstruttura, per comprendere in che modo ottimizzare tali proprietà per specifichi impieghi.

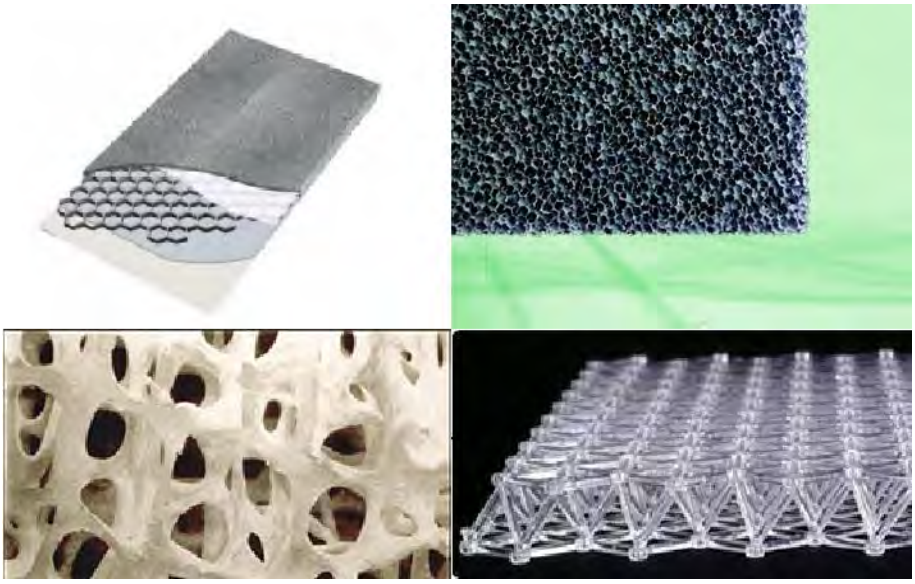

Figura 2.1 esempio di materiali cellulari: pannello honeycomb, spugna filtrante, osso trabecolare, struttura cellulare in polimero.

Una distinzione importante è tra materiali a cellula aperta, nei quali il volume vuoto di una cellula è in contatto con quello delle cellule adiacenti e materiali a cellula chiusa, nei quali il volume vuoto è isolato da una membrana di materiale. Tali differenza derivano dai metodi produttivi con cui i materiali sono ottenuti e generano comportamenti molto diversi tra loro: considerando ad esempio che i materiali a cellula chiusa utilizzati per l'assorbimento dell'energia cinetica in caso di impatto non sarebbero sostituibili con materiali a cellula aperta.

Le proprietà dei materiali cellulari, quali densità e modulo elastico, sono riportate in riferimento non al loro valore assoluto, ma al valore relativo ottenuto dividendo il valore riscontrato per il materiale cellulare per quello del materiale impiegato per la realizzazione della struttura.

In generale gli oggetti realizzati in materiale cellulare possono essere costituiti o solamente dalla mesostruttura, cioè la parte di struttura formata dalla ripetizione delle celle elementari oppure da mesostruttura ed esostuttura, dove quest'ultima indica il guscio superficiale che avvolge la mesostruttura [1]. Occorre specificare che la mesostruttura può essere costituita sia da celle aperte che da celle chiuse, ma di seguito ci si riferirà solamente a celle aperte, dato che sono di più semplice realizzazione e nelle applicazioni previste non si richiedono necessariamente le proprietà che contraddistinguono i materiali a cella chiusa.

Figura 2.2: componente che presenta mesostruttura a cella octet-truss [1].

La documentazione riguardante protesi ed innesti analizza le strutture cellulari con lo scopo di individuare quali meglio si prestino a rappresentare materiali schiumati, mentre

dove sono discusse le proprietà delle strutture a cellula aperta si considerano materiali non biocompatibili.

2.1. Tipi di celle.

Una cella è un insieme organizzato di elementi che si congiungono tra loro a formare una struttura organizzata. Parlando di celle aperte, si considerano elementi asta o trave e si dicono nodi i punti dove concorrono più elementi.
Ad oggi in letteratura sono proposte svariate tipologie di strutture cellulari, più o meno complesse dal punto di vista geometrico e contenenti un numero variabile di elementi. Si è riscontrato [2], per i materiali schiumati, che la resistenza è proporzionale a $\bar{\rho}^{1.5}$ nel caso in cui il comportamento degli elementi sia dominato da flessione, mentre è proporzionale a $\bar{\rho}$ nel caso in cui il comportamento degli elementi sia dominato da trazione-compressione, dove $\bar{\rho}$ rappresenta la desità relativa. Questo fatto spinge a comprendere quale sia l'effetto predominante nelle celle impiegate, se trazione-compressione o flessione. A tale fine si può impiegare il criterio di Maxwell esteso a strutture tridimensionali:

$$b - 3j + 6 \geq 0 \quad (2.1)$$

Dove b è il numero di elementi che costituisce la cella, mentre j è il numero di nodi della cella. Il criterio afferma che qualora la disequazione fosse verificata il meccanismo predominante sarebbe trazione-compressione, in caso contrario flessione.
Nel seguito saranno presentate alcune tipologie di celle di cui verranno descritte le proprietà geometriche e strutturali.

2.1.1. Cella Cubica semplice.

In questa tipologia di cella i nodi sono posti in corrispondenza dei vertici del cubo e gli elementi che li congiungono corrono lungo gli spigoli.

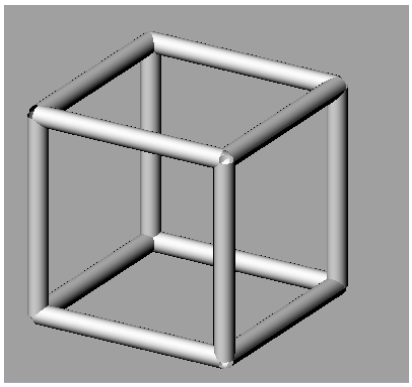

Figura 2.3 cella cubica semplice.

Questa è la tipologia più semplice tra le celle prese in considerazione, ha il pregio di ridurre il numero di elementi e nodi impiegati, così da minimizzare il costo computazionale in caso si considerasse di utilizzare tale cella in fase di elaborazione ed analisi. Il criterio di Maxwell indica che gli elementi sono dominati dalla flessione.
La cella cubica semplice può formare due tipologie di strutture cellulari

Struttura cubica semplice.

La Struttura cubica semplice, indicata come SC, è formata da celle cubiche semplici assemblate in maniera da far coincidere i vertici e gli spigoli delle facce adiacenti.

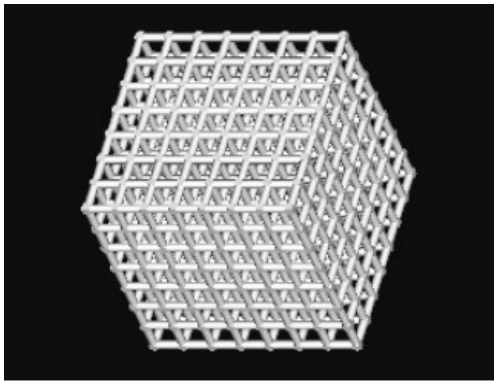

Figura2.4: struttura SC tratta da [3].

In [3] si vede che la struttura genera un buon livello di rigidezza specifica, ma si dimostra anche che i risultati ottenuti con le simulazioni FEM tendono a sovrastimare i

valori di resistenza riscontrati da prove di carico su un modello ottenuto con la tecnica SLS.

Struttura cubica semplice traslata.

Nota anche come TSC, la struttura è formata da celle cubiche semplici assemblate in maniera da far coincidere i vertici di una cella con il punto centrale dello spigolo di quella adiacente, come mostrato in figura.

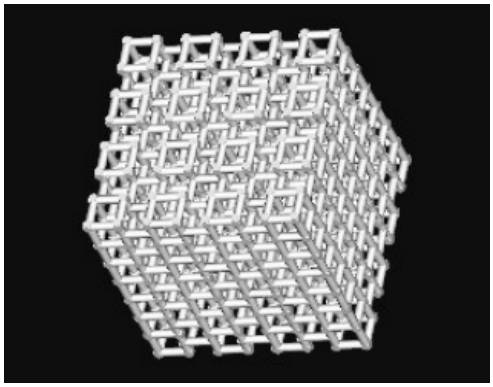

Figura 2.5: struttura TSC tratta da [3].

Questo assemblaggio porta ad ottenere una struttura fortemente anisotropa, in cui le proprietà sono fortemente determinate dalla direzione. La TSC è la cella che offre la miglior rigidezza specifica tra quelle affrontate in [3], riducendo rispetto alla SC lo scarto tra le previsioni ottenute dal FEM e i dati sperimentali.

2.1.2. Cella cubica a corpo centrato.

Detta anche BCC, la cella presenta rispetto alla cubica semplice un nodo aggiuntivo al centro della cella da cui partono gli elementi che lo collegano a tutti gli altri nodi della cella.

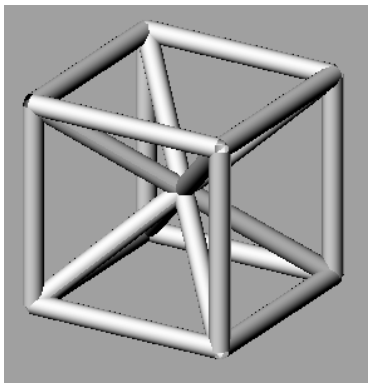

Figura 2.6: cella cubica corpo centrato BCC

Il criterio di Maxwell applicato alla struttura composta da celle BCC, sebbene dia ancora un valore negativo e quindi un comportamento dominato dalla flessione, restituisce un valore pari a -1. Questo valore si allontana poco da quello che si attende per una struttura dominata da trazione–compressione, facendo intuire che probabilmente in questa cella le due componenti di deformazioni saranno concorrenti. Tale aspetto meriterebbe un'indagine più approfondita. Con questa struttura si ottiene uno scarto tra i dati ottenuti dalla simulazione FEM e dalle prove sperimentali ridottissimo, a discapito di una modesta riduzione della rigidezza specifica, come indicato in [3]. Il crescere del numero di elementi per cella rende via via più oneroso il costo computazionale in fase di modellazione e analisi FEM e può rendere complesse le politiche per togliere il materiale in eccesso in fase di produzione.

2.1.3. Cubico a corpo centrato rinforzato.

Detta anche RBCC, si costruisce come la precedente con l'aggiunta di sei elementi che partono dal nodo presente al centro della cella e terminano al centro delle facce.

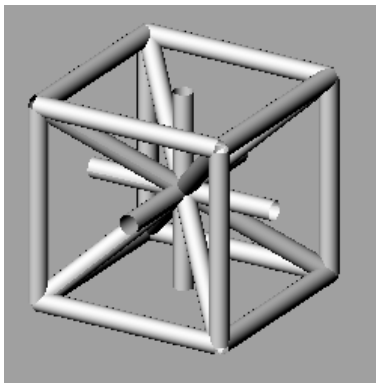

Figura 2.7: cella cubica a corpo centra
to rinforzato RBCC

Con 26 elementi e 9 nodi questa cella ha valore positivo per il criterio di Maxwell, indicando un comportamento dominato da trazione-compressione. La resistenza specifica offerta da questa cella è di poco inferiore a quella offerta dalla SC, ma a differenza di quest'ultima il calcolo FEM tende a sottostimare i risultati ottenuti con i campioni generati tramite SLS, come possiamo vedere in [3]. Questo garantirebbe di lavorare sempre in favore di sicurezza.

2.1.4. Gibson-Ashby.

Anche detta G-A, risulta di interesse perché gli elementi sono portati a lavorare a flessione in modo marcato, generando un valore nettamente negativo del criterio di Maxwell.

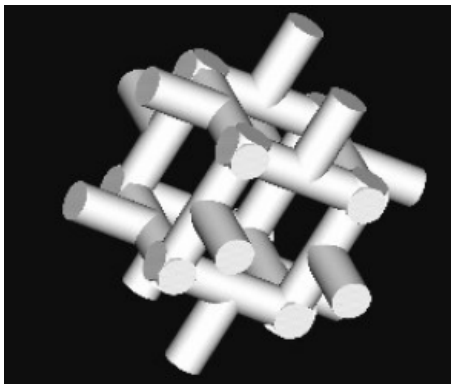

Figura 2.8: cella di Gibson Ashby, immagine
Tratta da [3]

La cella ha 24 nodi e 36 elementi e lo loro disposizione è tale da impedire la formazione di "colonne" di elementi che attraversano l'intero oggetto e garantendo un comportamento isotropo nelle tre direzioni principali della struttura.
Per le ragioni esposte, la struttura composta da questa cella dovrebbe comportarsi in modo potenzialmente simile all'osso trabecolare. Prove effettuate su tale struttura ma a discapito della sua resistenza specifica, che è la più bassa riscontrata in [3], inferiore ad 1/3 di quella ottenibile con la TSC.

2.1.5. Gibson-Ashby modificata.

Di seguito indicata come GAM questa cella è presentata in [4].

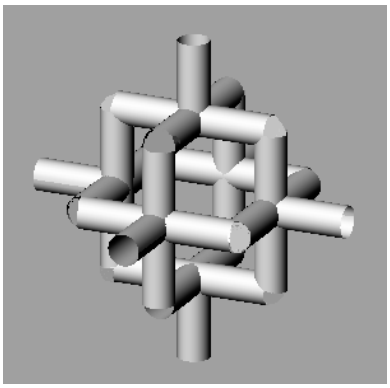

Figura 2.9: cella Gibson-Asby modificata.

Una volta assemblata la struttura si nota la presenza di un maggior numero di elementi di collegamento tra le celle, nonostante si mantengano le caratteristiche di comportamento a flessione e l'assenza di "colonne " di elementi che attraversano l'oggetto. Noto quanto presente in [5], in cui si evidenzia che al ridursi del numero di elementi si ha un crollo del modulo elastico relativo e della rigidezza specifica, si può sperare che questa tipologia riesca ad offrire i vantaggi della G-A, cioè l'analogia di comportamento con l'osso trabecolare e al contempo aumentare la resistenza specifica.

2.1.7. Octet truss.

La cella in questione è presentata in [2]. Dal punto di vista geometrico è costituita da un ottaedro sulle cui facce triangolari sono costruiti otto tetraedri, la struttura è meglio comprensibile dalla figura seguente.

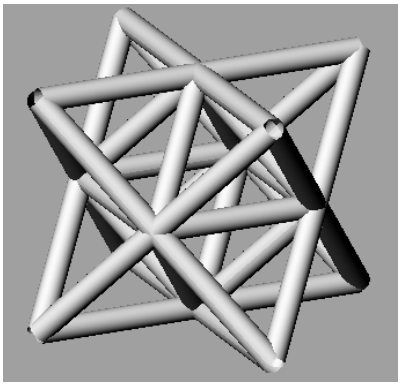

Figura 2.10: cella Octet Truss.

Ogni cella è caratterizzata da 36 elementi e 14 nodi, da cui si ottiene un valore nullo per il criterio di Maxwell, che colloca la cella OT tra le poche che risultano dominate da trazione-compressione

2.1.8. Wallach-Gibson.

Questa tipologia di celle è presentata in modo esteso in [6].

Figura 2.11: cella Wallach-Gibson.

La struttura in questione è costituita da 29 elementi e 9 nodi, che portano il criterio di Maxwell a definirla tra quelle dominate dalla trazione-compressione

2.2. Fabbricazione mediante il sistema *Selective Laser Sintering.*

Le tecniche *rapid prototyping*, dette RP, sono impiegate per la realizzazione di prototipi e operano sovrapponendo e facendo aderire l'un l'altro una successione strati di materiale. Si tratta di tecnologie *additive manufacturing* AM, dove appunto non c'è rimozione ma una successiva aggiunta di materiale per formare l'oggetto. Le tecnologie RP più comuni impiegano materiali quali foto-polimeri, come nella tecnica SLA o lamine di materiale, come nella tecnica LOM, che sono adatti per la realizzazione di prototipi, ma non altrettanto per la realizzazione di prodotti finiti. Infatti queste tecniche possono realizzare oggetti impiegando un limitato numero di materiali, spesso polimeri che presentano ridotte caratteristiche di resistenza meccanica, oppure che presentano problemi di adesione tra gli strati e l'impossibilità di realizzare parti con pareti sottili sufficientemente resistenti.

L'evoluzione tecnologica ha consentito lo sviluppo di processi di tipo additivo in grado di superare le limitazioni descritte, aprendo di fatto la strada verso il *rapid manufacturing* RM, un processo per la realizzazione di prodotti finiti che impiega tecniche RP. Tra le tecniche più promettenti c'è la SLS o *Selective Laser Sintering*, basata sulla sinterizzazione mediante laser di polveri di materiale per la formazione di strati che aderiscono reciprocamente.

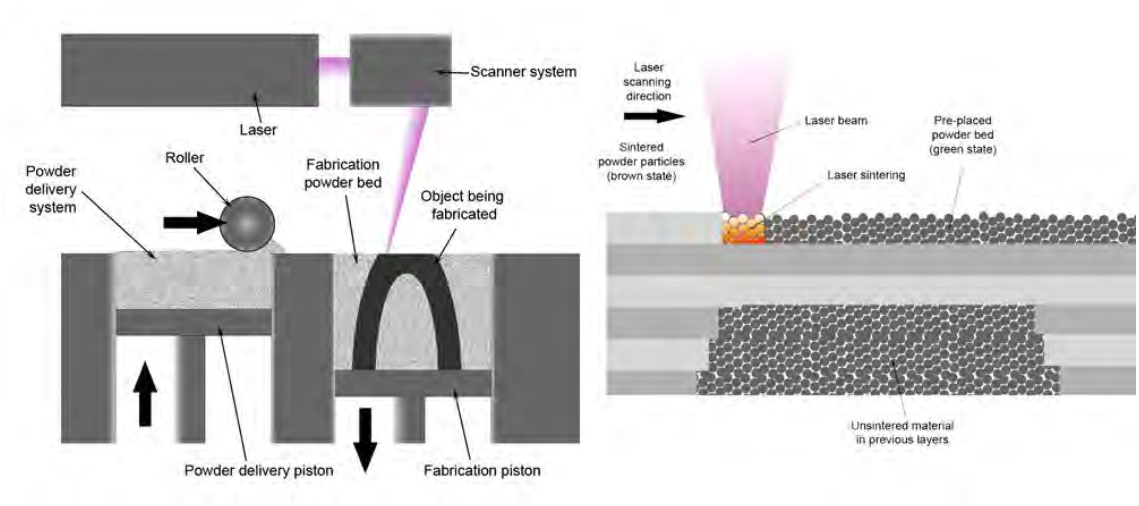

Figura 2.12: schema di funzionamento del processo Selective Laser Sintering.

Il processo di sinterizzazione può portare a fusione parziale oppure a fusione totale del materiale, a seconda dell'intensità con la quale il raggio laser colpisce le polveri e della velocità di avanzamento. Quando la tecnica è impiegata nel campo del RM si ricerca il *full melting* del materiale intercettato dal laser per ottenere densità prossime a quelle ottenibili coi metodi di produzione canonici.

Nonostante la fusione totale, questa tecnologia porta inevitabilmente ad ottenere un materiale anisotropo a causa del processo di fusione discontinuo a cui sono soggetti gli strati si materiale. Il tempo che trascorre perché il laser ripassi nello stesso punto, incidendo però un nuovo strato di materiale, è detto *delay time*. Questo dipende dalla dimensione dell'area da portare a fusione col laser e poiché che al ridursi del volume di materiale da sinterizzare si riduce anche l'area per singola passata, nella realizzazione di strutture cellulari il *delay time* e quindi l'anisotropia conseguente possono essere minimi. L'impiego di percorsi incrociati per il laser, tecnica detta *cross scanning*, e l'adozione di un sistema per il riscaldamento del letto di polveri possono limitare l'anisotropia nel materiale.

Gli elevati gradienti di temperatura e il repentino cambiamento di densità, da 50% della polvere al quasi 100% del fuso, generati dal *full melting* causano tensioni residue e conseguenti deformazioni sul prodotto. Tali fenomeni possono essere limitati grazie ad una opportuna regolazione dei parametri di controllo del processo e con il riscaldamento del letto di polvere. Quest'ultima procedura può compromettere il riciclo del materiale

non sinterizzato a causa del degrado che causa nei confronti del materiale non ancora sinterizzato.

Il processo di consolidamento del materiale fuso è guidato dalla gravità e dalla capillarità, la viscosità allo stato liquido del materiale impiegato è quindi un parametro essenziale per garantire l'idoneità per l'impiego nella fabbricazione con tecnica SLS.

Tra i materiali di uso corrente si annoverano i Poliammidi, PA, tra i polimeri, dato il loro basso costo e il titanio, tra i materiali metallici, date le sue proprietà meccaniche elevate associate alla biocompatibilità.

Nel caso dei polimeri è necessario distinguere tra amorfi e semicristallini. Questi ultimi hanno una temperatura di fusione Tm ben definita, superata la quale avviene un repentino crollo della viscosità e perciò garantiscono un processo di consolidazione che permette di ottenere basse porosità e quindi proprietà meccaniche paragonabili a quelle fornite da manufatti ottenuti con le tecniche convenzionali. Gli aspetti negativi del repentino passaggio dallo stato liquido al solido sono la presenza di elevato ritiro e conseguente distorsione del prodotto finito. I polimeri amorfi, non mostrano il passaggio di stato da solido a liquido, ma variano con continuità la propria viscosità in funzione della temperatura, senza però arrivare ai valori di scorrimento che caratterizzano i polimeri semicristallini. Per tali ragioni il processo di consolidamento per i polimeri amorfi porta ad ottenere maggiore porosità e un solido con minori proprietà meccaniche rispetto a quelli che si otterrebbero utilizzando polimeri semicristallini. Il passaggio graduale tra i due stati limita però il ritiro e i problemi di tensione residua, suggerendo pertanto l'impiego dei polimeri amorfi nelle applicazioni in cui l'estetica sia fondamentale.

I vantaggi di tale tecnologia sono svariati e riguardano sia aspetti economici, legati all'ottimizzazione di impiego del materiale e alla riduzione dei tempi di produzione, che alla possibilità di realizzare forme con morfologia complessa senza aumentare la complessità del processo produttivo.

3. Introduzione agli algoritmi genetici.

Gli algoritmi genetici (AG), proposti nel 1975 da J.H. Holland[7], sono un modello computazionale idealizzato dell'evoluzione naturale Darwinista. Per esporre tale teoria è necessario chiarire il significato di alcuni termini specifici della biologia: fenotipo e genotipo. Il fenotipo non è altro che l'insieme delle caratteristiche fisiche di un individuo, mentre il genotipo è il patrimonio genetico di un individuo. Il fenotipo è determinato sostanzialmente dall'invisibile patrimonio genetico del soggetto. Le caratteristiche fisiche, il fenotipo, dettano le possibilità e i limiti dell'interazione dell'individuo con l'ambiente in cui vive.

La teoria dell'evoluzione naturale è basata su due principi: la variazione genetica e la selezione naturale. Il primo afferma che, affinché la popolazione possa evolvere, gli individui che la costituiscono debbano avere una ricca varietà genetica, in modo da generare individui con nuovi fenotipi e quindi con differenti caratteristiche di adattamento all'ambiente. Il principio di selezione naturale afferma invece che gli individui meglio adattati hanno più alte probabilità di riprodursi e trasmettere il prorprio patrimonio genetico alle nuove generazioni.

La varietà del genotipo è ottenuta sostanzialmente in due modi: un processo combinatorio dei geni, grazie ai diversi apporti dei genitori nell'ambito della riproduzione sessuale e dalle mutazioni geniche casuali. Ad ogni generazione il cosiddetto *pool* di geni sul quale va ad agire la selezione naturale è perciò in continuo cambiamento ed essendo le variazioni di fenotipo che apportano contributi migliorativi all'adattamento di tipo additivo, i continui miglioramenti apportati si accumulano nell'arco di tempi lunghissimi dando origine a grossi cambiamenti. Questo processo è chiamato evoluzione additiva e la conseguenza di questo processo è la tendenza a generare individui sempre meglio adattati all'ambiente.

3.1. Algoritmo genetico di base.

Sono tecniche euristiche di calcolo *general purpose*, relativamente nuove, ispirate alla meccanica dell'evoluzione naturale. Gli AG vengono applicati in un ampio spettro di

problematiche: da problemi di natura prettamente controllistica per gasdotti, altiforni e la guida di missili terra-aria, a problemi di natura ottimizzatoria, come i problemi di progettazione di turbine e parti aerodinamiche di velivoli, oppure alla modellazione di mercati finanziari telematici. In alcuni casi gli AG vengono utilizzati in successione ad altri algoritmi, in tal caso i primi sono pensati per ottenere soluzioni ammissibili, mentre gli algoritmi genetici si concentrano solo sulla ricerca di ottimi.

Gli AG non hanno ancora avuto una elevata diffusione in problemi di ricerca operativa e di *decision making* e tuttora vi è in ambito scientifico una certa riluttanza nel loro utilizzo. Ciò è dovuto in parte a ragioni storiche e in parte a difficoltà implementative.

I problemi di natura implementativa derivano dalla stretta correlazione fra la metodica algoritmica e fenomeno naturale da cui prendono ispirazione, di per se alquanto complesso sia in termini fenomenici che nella terminologia.

Dato un problema da risolvere, gli AG considerano le possibili soluzioni come i singoli individui della popolazione e la performance della soluzione come il grado di adattamento dell'individuo all'ambiente, quest'ultima la definizione di fitness di un individuo in relazione all'ambiente. L'algoritmo fa incominciare l'evoluzione fornendo, in base al principio di selezione naturale, l'opportunità agli individui di accoppiarsi. La progenie, che ha una struttura genetica data dalla combinazione dalle stringhe di bit costituenti il patrimonio genetico dei genitori, va a formare la nuova generazione in sostituzione della vecchia. La ricerca della soluzione al problema procede con una serie di generazioni in cui ogni individuo contribuisce alla successiva generazione in proporzione alla sua fitness.

L'evoluzione della popolazione di stringhe è casuale, ma il suo tasso di miglioramento è fortemente superiore a quello di una semplice *random walk* [7] dato che, in analogia con l'evoluzione biologica, gli AG impiegano l'esperienza passata per effettuare una ricerca efficiente. Lavorando con popolazioni di soluzioni, parallelismo intrinseco, guidate dal principio di selezione naturale, gli AG favoriscono la diffusione nella popolazione dei miglioramenti conseguiti ai passi evolutivi precedenti.

3.1.1. Rappresentazione genetica di un problema

Per poter applicare l'algoritmo genetico, occorre anzitutto codificare numericamente le soluzioni e per fare ciò viene utilizzata una stringa di lunghezza costante formata da geni, solitamente con codifica binaria. La funzione di fitness è invece la misura numerica della "bontà" di una soluzione, la fitness

Come esempio per chiarire i concetti esposti supponiamo di dover determinare il massimo della funzione:

$Y = X+| \sin(32\ X) |$ con $0 \leq X \leq \pi$

Tale massimo non può essere determinato per via analitica tramite l'annullamento della derivata della funzione a causa della presenza del valore assoluto del seno. Può però essere trovato facilmente dall'algoritmo genetico considerando stringhe binarie a rappresentazione del valore di X. Essendo quest'ultimo compreso nell'intervallo 0-3.141519 e considerando una discretizzazione con cento suddivisioni, occorrono trecentoquattordici valori diversi, con stringhe di 9 bit, dato che con $2^9 = 512$ valori rappresentabili mentre $2^8 = 256$. La funzione di fitness coincide ovviamente con la stessa funzione Y.

Come ulteriore esempio si consideri un labirinto dato e un robot che debba percorrerlo dall'entrata all'uscita in un massimo di cento passi. I movimenti possibili di ogni passo siano avanti A, indietro I, sinistra S, destra D), così che una generica sequenza potrebbe essere:

DSSAADSI ……D

Codificando le possibili mosse con due bit, una stringa generica rappresentante le cento mosse avrà duecento bit. La fitness di ciascuna stringa può essere misurata dalla distanza fra l'uscita del labirinto e il punto in cui si troverebbe il robot alla fine della sequenza.

3.2. Galapagos, il risolutore genetico di Grasshopper.

L'ambiente di lavoro di Grasshopper offre la possibilità di usufruire di un risolutore genetico in grado di trovare il minimo o il massimo di una data funzione. Perciò, l'approccio corretto nell'uso di Galapagos prevede innanzitutto una definizione accurata

del problema, una codifica delle variabili in gioco e la determinazione della funzione obbiettivo da massimizzare o minimizzare.

3.2.1. Descrizione dei passi compiuti dal risolutore genetico.

1) **Creazione della g[0].**

   Noto il numero di individui della popolazione, il primo passo da compiere è popolare lo spazio di ricerca con delle soluzioni determinate in modo casuale che vanno a costituire la generazione iniziale g[0].

2) **Valutazione della funzione obbiettivo.**

   Per ogni individuo della popolazione viene determinato il valore della funzione obbiettivo. Per la generazione g[0] presumibilmente il valore sarà basso, ma non è da escludere la presenza di soluzioni con elevata fitness.

3) **Creazione della nuova generazione.**

   Partendo dalla generazione attuale, genericamente denominata g[n], si vuole costruire g[n+1] a numerosità della popolazione invariata.

   Gli individui di g[n+1] possono essere o nuovi individui generati a partire da quelli di g[n] oppure individui di g[n] che superano il passaggio generazionale mantenendosi anche in g[n+1]. La generazione di un nuovo individuo segue i passi descritti di seguito:

   - Selezione.

     Viene scelto il primo elemento della coppia in funzione del valore della fitness dell'elemento.

   - Accoppiamento.

     Considerando l'elemento selezionato e una platea di possibili partner, conoscendo lo schema genetico degli elementi si calcola la distanza di Hamming tra gli elementi delle coppie considerate. L'individuo prescelto per formare la coppia è quello con il requisito di distanza genetica maggiormente compatibile con le richieste.

     Per definire la distanza di Hamming si considerino gli elementi 1100 e 1001, la distanza genetica è d(1100;1001)=2, mentre per 1100 e 1101 è d(1100;1101)=1.

La distanza genetica obbiettivo è definita per mezzo *dell'imbreeding factor*, che può variare nell'intervallo +100%; -100%, dove +100% identifica la ricerca di tra individui identici, mentre -100% tra individui totalmente diversi.

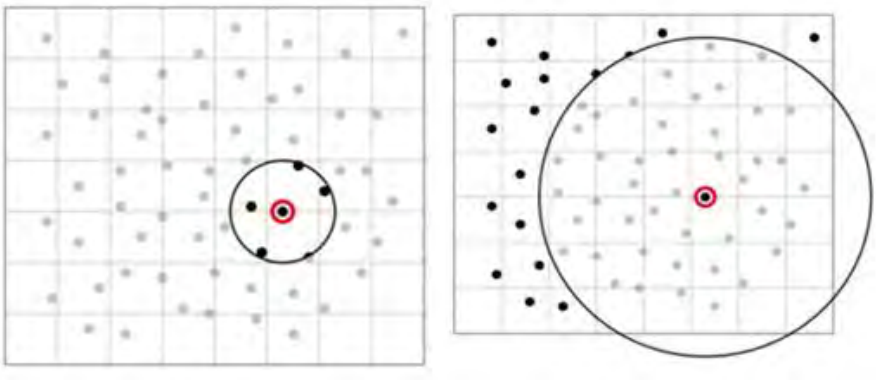

Figura 3.1: selezione degli individui geneticamente vicini e selezione di quelli geneticamente lontani.

Comportamenti incestuosi portano allo sviluppo gruppi sub-ottimali con una varietà genetica limitata e non utile a raggiungere l'ottimo globale. Mentre comportamenti zoofili non consentono lo sviluppo di gruppi forti.
Una scelta equilibrata dell'*Imbreeding Factor* consente di evitare sia l'incesto, per valori vicini a +100% che la zoofilia, per valori vicini a -100%.

- Scambio del patrimonio genetico.
  Lungo la stringa di bit costituente il patrimonio genetico degli individui della coppia viene scelta una posizione di incrocio. Il patrimonio genetico il nuovo individuo sarà identico al genitore 1 fino al punto di incrocio, mentre sarà quello del genitore 2 dopo il punto di incrocio o viceversa.

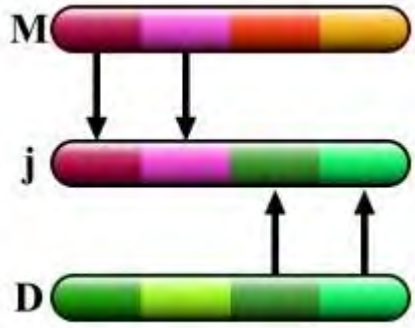

Figura 3.2: scambio del patrimonio genetico di M e D per la generazione di j.

- Mutazione.
  Fenomeno tale per cui, con frequenza piuttosto bassa, viene indotta la modifica casuale di uno o più geni della stringa.

I passi 2) e 3) vengono ripetuti finché non sopraggiungono le condizioni per interrompere l'algoritmo.

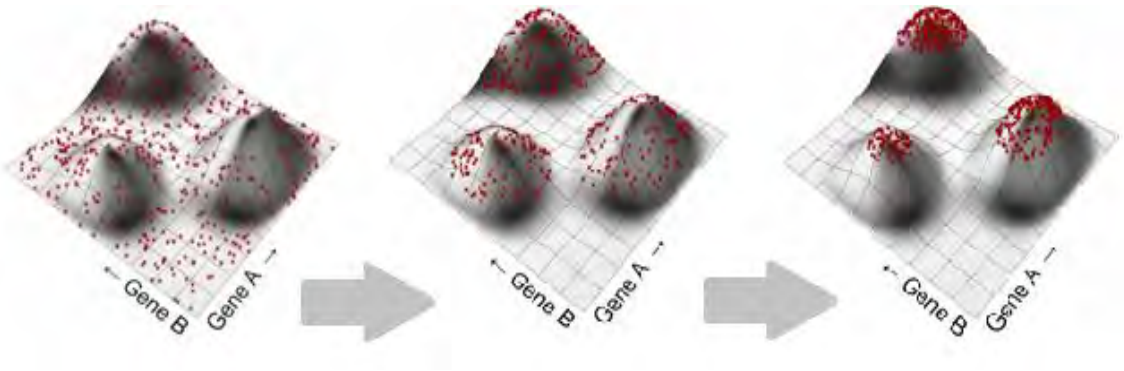

Figura 3.4: evoluzione delle soluzioni in uno spazio bi-dimensionale

L'immagine ha lo scopo di mostrare in maniera semplificata la possibile evoluzione delle soluzioni presenti nello spazio di ricerca al susseguirsi delle generazioni. Il fenomeno della risalita degli individui verso i picchi è indice dell'efficacia dell'algoritmo.

3.2.2. La finestra di dialogo di Galapagos.

La finestra di dialogo di Galapagos è composta da tre menù principali: *Options, Solvers, Record*. Il primo contiene tutte le impostazioni del risolutore ed è organizzato a sua volta in tre raggruppamenti: *Generic*; *Evolutionary Solver*; *Annealing Solver*. Il menù *Solvers* contiene invece le finestre di dialogo con il risolutore, in particolare per la scelta del tipo di solutore da utilizzare tra il genetico e l'*annealing*, per l'avvio del risolutore, oltre che molte altre schermate dove viene riprodotto l'andamento del genoma al variare delle iterazioni. Il menù *Record* non è altro che il registro di tutte le attività svolte dal risolutore al passare delle iterazioni.

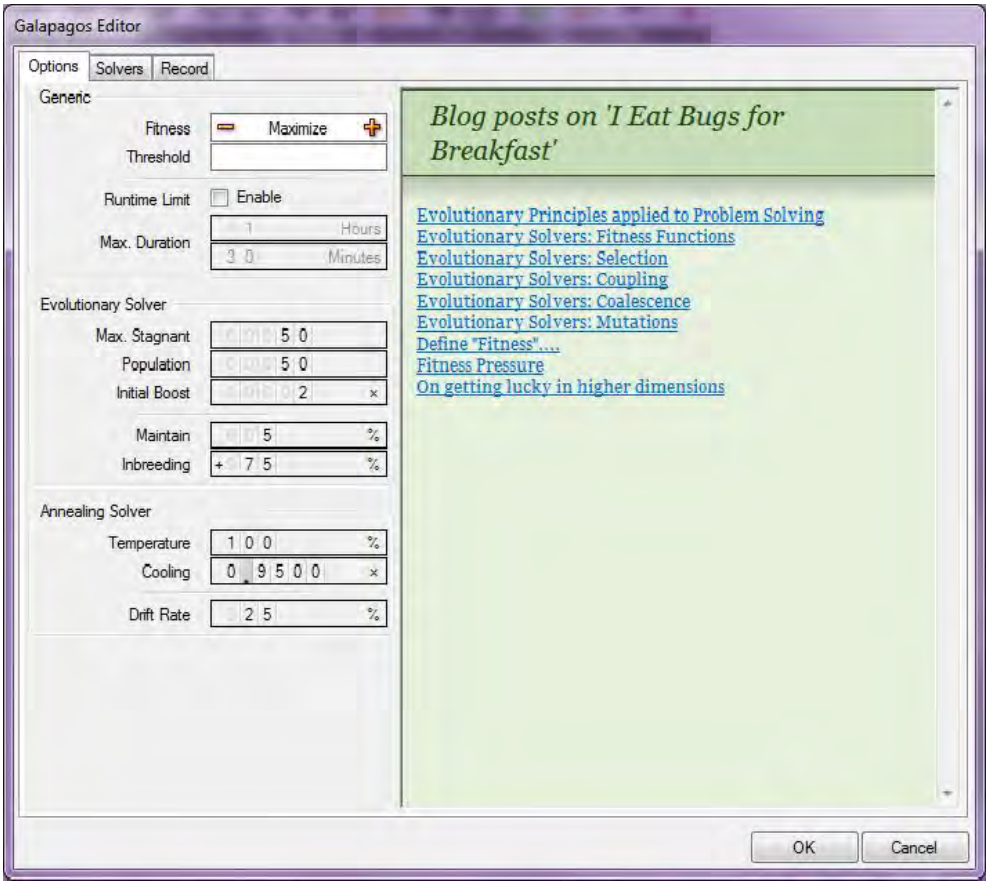

Figura 3.5: finestra di dialogo di Galapagos.

La sottocategoria *Generic* del menù *Options* contiene le opzioni: *fitness*, in cui scegliere se minimizzare o massimizzare la funzione obbiettivo, *Threshold*, in cui fissare il valore di soglia che una volta raggiunto dalla funzione permette all'algoritmo di fermarsi e in ultimo l'opzione *Runtime limit* che permette di fissare un tempo massimo per l'esecuzione delle iterazioni.

La sottocategoria *Evolutionary Solver* racchiude le opzioni relative al risolutore genetico. Tra queste *Max Stagnant* rappresenta il numero di iterazioni riscontrate senza variazioni del genoma migliore dopo le quali l'algoritmo viene interrotto. *Population* indica invece la popolosità della popolazione che evolve, mentre *Initial Boost* è il fattore di moltiplicazione che definisce la popolosità della generazione g[0] pari a *Population* x *Initial Boost*.

Mantain indica invece la percentuale della popolazione che può compiere il salto generazionale dalla generica g[n] alla g[n+1]. Gli individui che possono compiere il salto generazionale sono scelti in base al valore della funzione obbiettivo. Infine per quanto concerne invece *l'Imbreeding* vale quanto affermato nel paragrafo precedente relativamente all'*Imbreeding Factor*. I valori presenti nella figura 3.5 sono quelli impostati di default nel software.

4. Strumenti e metodi.

Le procedure presentate sono ottenute con l'utilizzo dei software Rhinoceros5, Grasshopper e Karamba. Il primo è il software CAD utilizzato per la rappresentazione dei modelli solidi della struttura, Grasshopper e Karamba sono invece dei *Plug-in* di Rhinoceros5. Grasshopper è l'applicativo per il disegno e la progettazione parametrica, mentre Karamba è l'applicativo di Grasshopper per l'analisi strutturale FEM. Grasshopper e Karamba operano nel medesimo ambiente, quello di Grasshopper, che si presenta costituito da un *canvas* o tela sulla quale importare dei blocchi input-output che eseguono operazioni su liste di dati, rappresentanti la geometria o altre caratteristiche del modello.

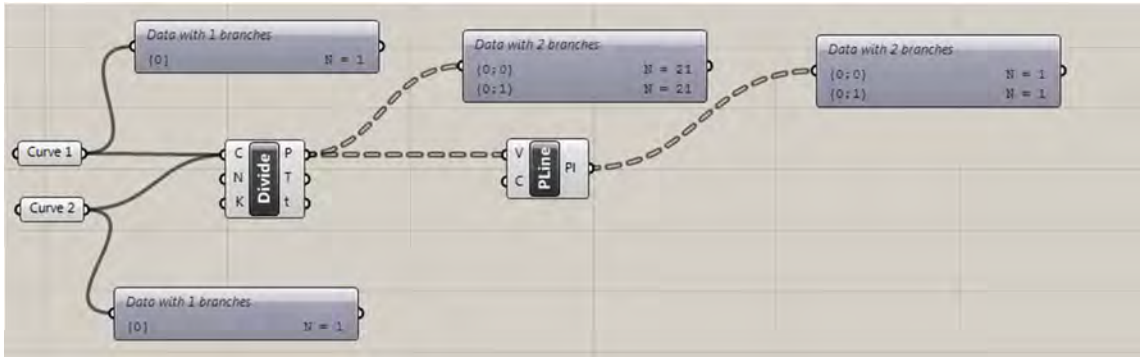

Figura 4.1: ambiente Grasshopper.

I blocchi richiedono in input, oltre ai dati da elaborare, i parametri di controllo dell'operazione forniti dall'utente. Il passaggio di dati da un blocco all'altro avviene stabilendo dei collegamenti tra i blocchi per mezzo di cavi che trasportano flussi di dati dall'output del blocco che precede all'input del successivo, come mostrato in figura 4.1. Non sono possibili collegamenti da input verso input in quanto non conterrebbero dati.
L'ambiente, oltre alla miriade di blocchi che effettuano operazioni geometriche, dispone anche di numerosi strumenti supplementari, come la già citata l'applicazione Karamba per l'analisi strutturale o altri strumenti che offrono la possibilità di creare blocchi che eseguono script in linguaggio Python creati dall'utente. La possibilità di integrare

strumenti che eseguono diversi tipi di analisi e operazioni all'interno dello stesso ambiente, utilizzando una base di dati condivisa tra loro, rende Grasshopper uno strumento di progettazione estremamente flessibile. Oltre alla flessibilità, l'ambiente offre anche il vantaggio di automatizzare la procedura, velocizzando di molto l'esecuzione delle singole operazioni, sia di disegno che di calcolo, aprendo inoltre la strada dell'utilizzo di procedure iterative.

Di seguito verranno presentate le procedure sviluppate utilizzando gli stumenti descritti, partendo con la procedura di tipo iterativo per l'ottimizzazione delle beam, poi presentando quella da questa derivata, ma che consente di considerare anche la presenza degli elementi shell e infine la procedura basata sull'algoritmo genetico per l'ottimizzazione delle *beam*

4.1. Procedura iterativa per l'ottimizzazione delle *beam*.

Questa procedura riguarda la progettazione e l'ottimizzazione automatica del cosiddetto mesoscheletro della struttura cellulare. Come già illustrato nel capitolo 2, con mesostruttura si indica la parte della struttura costituita dalla ripetizione di celle elementari che costituisce la parte interna dell'oggetto. I passi seguiti dalla procedura per la progettazione e ottimizzazione automatica sono mostrati nella figura seguente.

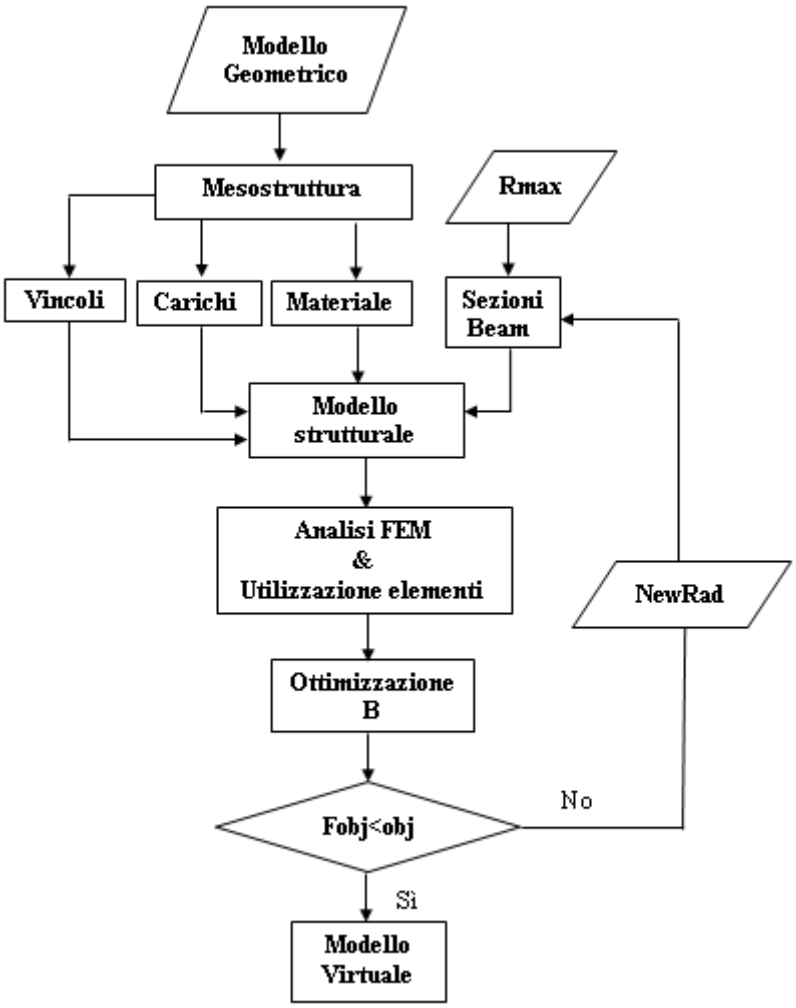

Figura 4.2: schema della procedura iterativa di ottimizzazione beam

Nello schema i blocchi a forma di parallelogramma segnalano input, quelli rettangolari operazioni sui dati, mentre quelli romboidali verifiche. Affinché la procedura possa effettuare l'ottimizzazione strutturale in modo automatic oè necessario che sia iterativa e agisce variando opportunamente la dimensione delle sezioni degli elementi che la compongono. La procedura continua nel suo processo fino a che la verifica sulla funzione obbiettivo Fobj, posta nel blocco romboidale, non è soddisfatta. Quando la procedura si interrompe è automaticamente fornita la configurazione geometrica della struttura rappresentata in Rhinoceros5 come modello solido

Di seguito si descrivono nel dettaglio i blocchi riportati nello schema

4.1.1. Modello geometrico.

Il modello geometrico è il modello solido della parte costituito dalla superficie esterna che racchiude un volume. Attualmente la procedura accetta modelli solidi da polisuperfici chiuse e non tratta solidi rappresentati da *mesh* superficiale chiusa. Ciò non garantisce la necessaria flessibilità di lavoro dato che è molto frequente avere a che fare con modelli solidi da *mesh* chiuse, come ad esempio quelli in formato .stl.

4.1.2. Mesostruttura.

La struttura di riempimento del volume racchiuso dalla superficie del solido viene ottenuta per mezzo della ripetizione di celle elementari collegate tra di loro e prende il nome di mesostruttura.

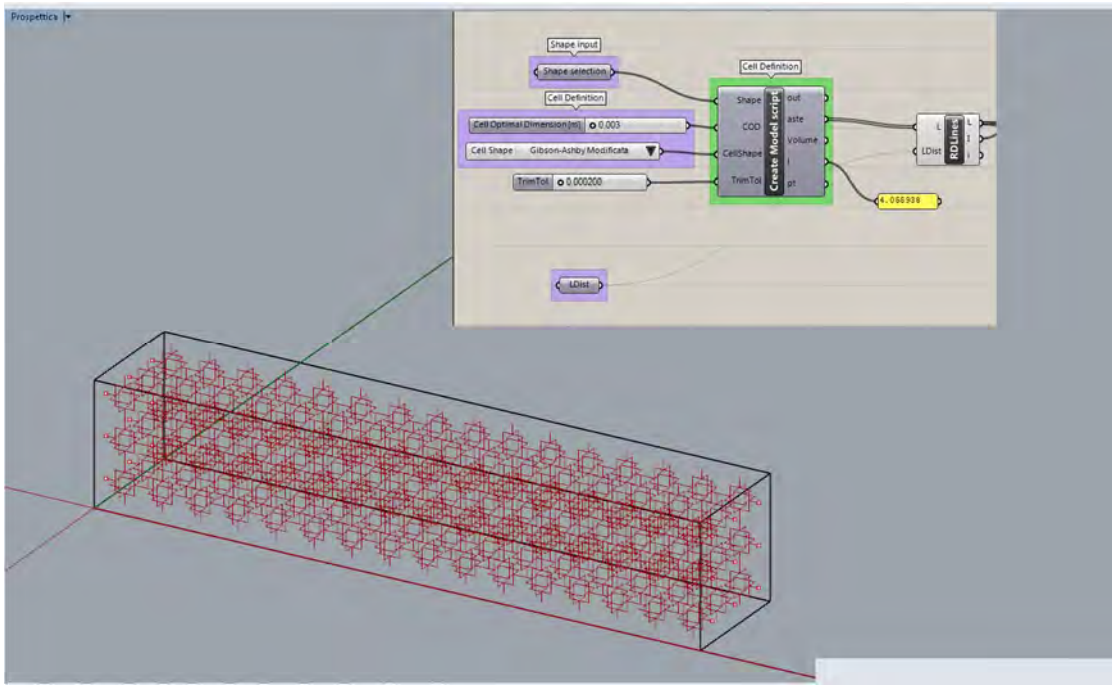

Figura 4.3: rappresentazione della mesostruttura tramite linee.

In figura 4.3 è riportata anche la parte di procedura per il disegno della geometria schematica della mesostruttura tramite linee. Il componente fondamentale è il blocco chiamato *Create Model Script*, costituito da uno script in linguaggio Python, a cui sono collegati gli input *Shape*, *CellDimension*, *CellShape* che definiscono rispettivamente il modello solido, la dimensione della cella e la forma della cella. Lo script *Create Model Script* esegue le seguenti operazioni fondamentali:

1. determinazione della *BoundingBox*che contiene il solido;
2. determinazione dei vertici di partenza per la costruzione delle singole celle;
3. creazione delle celle di forma voluta;
4. eliminazione delle linee che fanno parte della *BoundingBox*, ma che non intersecano il volume del solido;
5. *trim* delle linee che sporgono rispetto al volume del solido.

Per maggiori dettagli sulle singole operazioni si rimanda all'appendice A contenente il codice di *Create Model Script.*

Una volta ottenuta la geometria schematica della mesostruttura costituita dalle linee del modello, è possibile assegnare a queste le proprietà dell'elemento strutturale *beam*. Gli elementi *beam* sono elementi finiti monodimensionali, ma di applicazione spaziale, costituiti da almeno due nodi posti alle estemità dell'elemento. Sono particolarmente adatti a rappresentare componenti dove una dimensione è prevalente sulle altre due e da questo punto di vista, la sua geometria corrisponde all'asse baricentrico del componente che si vuole rappresentare. Dal punto di vista meccanico il componente sopporta azione assiale, di taglio, flessione e torcente. L'assegnazione della proprietà di elemento voluta è fatta tramite il componente *LineToBeam* di Karamba mostrato in figura seguente.

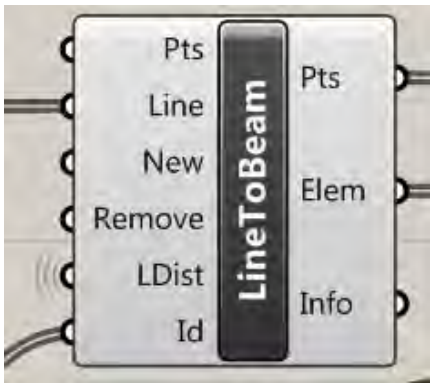

Figura 4.4: componente LineToBeam di Karamba

L'*input* Line è la lista contenente le linee che compongono lo schema della mesostruttura. Ldist è la tolleranza per entro cui considerare due entità vicine una unica. Gli *output* sono la lista degli elementi strutturali assegnati alle linee, nella lista Elem, ed i nodi della struttura Pts.

4.1.3. Vincoli, Carichi, Materiale e Sezioni Beam.

Per l'assegnazione dei vincoli della mesostruttura è necessario l'impiego del componente "Support" di Karamba. Questo richiede in input le coordinate dei punti della da vincolare e permette di definite gli spostamenti assoluti da bloccare per ogni nodo vincolato

Figura 4.5: componente Support di Karamba.

I carichi sono definiti dal componente "PointLoad" di Karamba, il quale richiede sia le coordinate dei punti da caricare che i vettori forza che vanno ad agire sui nodi prescelti.

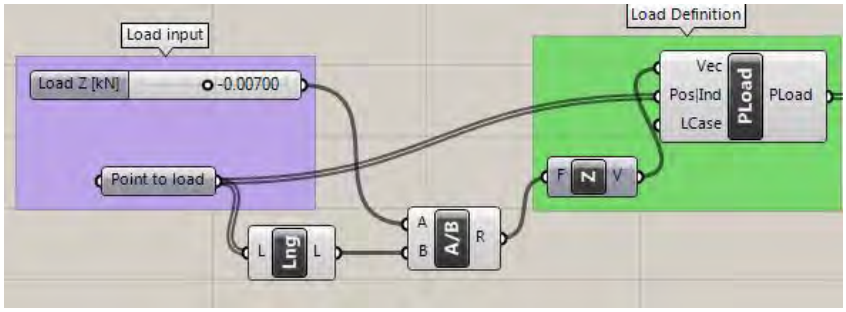

Figura 4.6: componente PointoLoad di Karamba.

Nell'esempio riportato in figura si vede che il carico totale di 7N è suddiviso tra i nodi selezionati contenuti in *Point to load*.

La definizione del materiale avviene sempre con un componente "MaterialProperties" di Karamba. Gli *input* per il componente sono mostrati in figura 4.7.

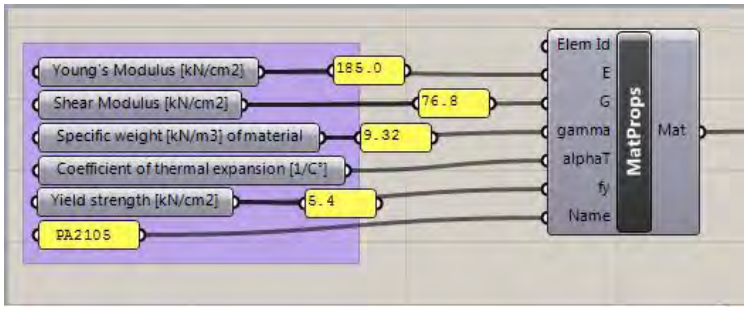

Figura 4.7: componente MaterialProperties di Karamba.

In figura è riportata la definizione delle proprietà meccaniche del materiale PA2105, un polimero impiegato nella produzione tramite tecnica SLS *Selective Laser Sintering.*

Ogni forma di sezione trasversale delle *beam* è definita da un apposito componente di Karamba che permette di impostare le dimensioni volute. Per ottenere una struttura semplice ed affidabile si è scelto di utilizzare sezioni circolari piene

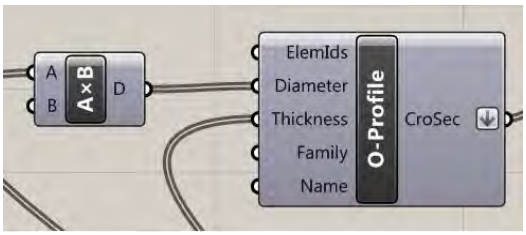

Figura 4.8: componente O-Profile di Karamba.

Questa geometria è ottenuta per mezzo del componente “O-Profile” che determina sezioni sia piene che vuote in base allo spessore impostato. In figura 4.8 si vedono gli ingressi Diameter, e Thickness che definiscono dimensioni e spessore delle sezioni.

4.1.4. Modello Strutturale.

Per la creazione del modello strutturale occorre solamente unire i dati elaborati tramite i componenti “LineToBeam”, “O-Profile”, “Support”, “PointLoad” e “MaterialProperties” utilizzando il componente “Assemble” di Karamba.

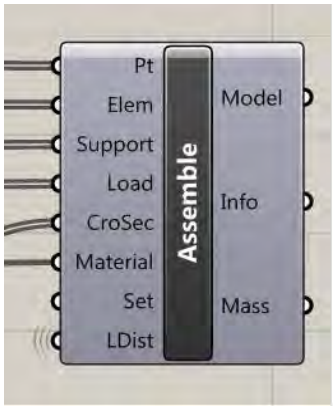

Figura 4.9: componente Assemble di Karamba.

L’output del componente è il modello strutturale costituito nella maniera illustrata e pronto per l’analisi FEM contenuto nell’*output* Model.

4.1.5. Analisi FEM &Utilizzazione elementi.

L'uscita Model di "Assemble" diventa *input* per "Analyze", componente che effettua l'analisi FEM. della struttura. Da quest'ultimo, oltre al modello analizzato, vengono fornite altre informazioni e grandezze d'interesse in ambito strutturale come il massimo spostamento dei nodi della struttura e l'energia elastica immagazzinata.

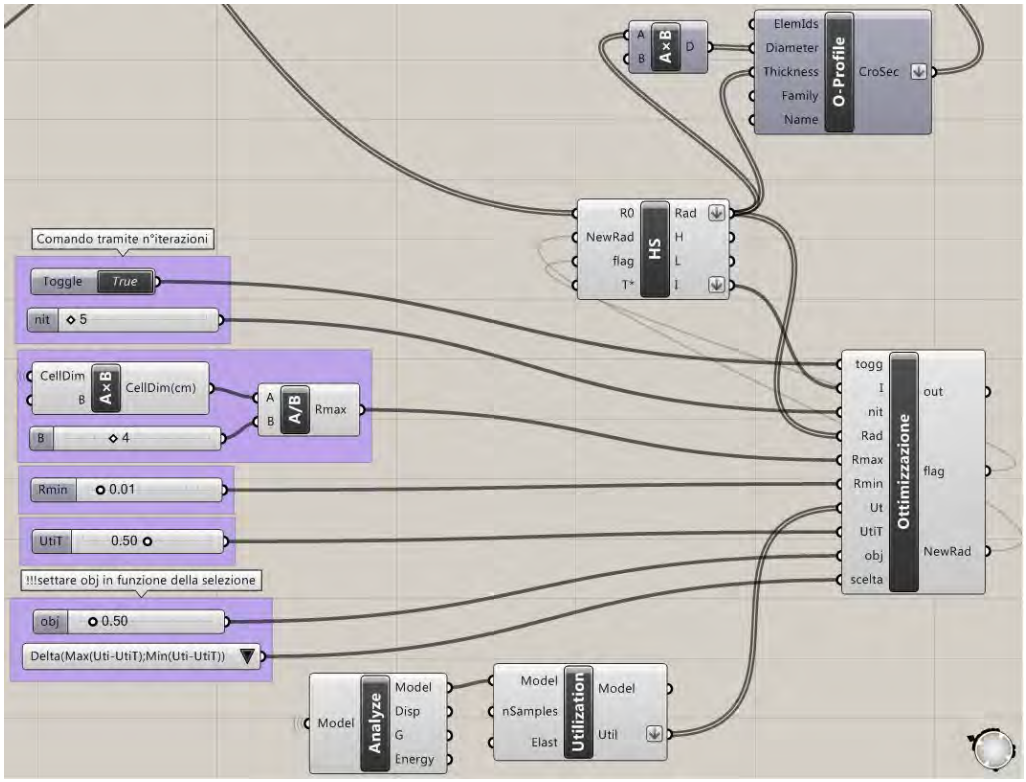

Figura 4.10: interazione tra i componenti Analyze, Utilization e lo script Ottimizzazione.

L'utilizzazione degli elementi è definita dal rapporto tra la forza effettivamente agente sul nodo e quella massima sopportabile dalla sezione considerata per l'elemento. Quest'ultima è la forza che determinerebbe una distribuzione teorica di tensioni nella sezione tale da causare il raggiungimento della tensione di snervamento per il materiale. Per quanto detto, nel caso in cui l'utilizzazione dell'elemento superasse l'unità, significa che la sezione impiegata ha superato la tensione di snervamento. Perciò, al fine di garantire un sufficiente livello di sicurezza, si considererà come limite superiore dell'utilizzazione accettabile per un elemento il 50%

Per un elemento beam l'utilizzazione che viene assegnata è la massima riscontrata sui nodi che lo compongono. Il calcolo dell'utilizzazione segue una procedura che porta alla determinazione degli indici fb e fs per i nodi dell'elemento e che l'assegna pari al massimo valore degli indici calcolati sui nodi dell'elemento in esame. Per implementare la procedura devono essere note le sei caratteristiche della sollecitazione (N, Vy, Vz, Mt, My, Mz) per ogni nodo della struttura. Gli indici fb e fs sono calcolati nel modo seguente:

$$fb = \frac{N}{N_{rd}} + \left|\frac{My}{My_{rd}}\right| + \left|\frac{Mz}{Mz_{rd}}\right| + \left|\frac{Mt}{Mt_{rd}}\right| \text{ per } N>0 \text{ (trazione)} \quad (4.1)$$

$$fb = -\frac{N}{N_{brd}} + \left|\frac{My}{My_{rd}}\right| + \left|\frac{Mz}{Mz_{rd}}\right| + \left|\frac{Mt}{Mt_{rd}}\right| \text{ per } N<0 \text{ (compressione)} \quad (4.2)$$

$$fs = \left|\frac{Ty}{Ty_{rd}}\right| + \left|\frac{Tz}{Tz_{rd}}\right| + \left|\frac{Mt}{Mt_{rd}}\right| \quad (4.3)$$

Le componenti della sollecitazione contraddistinte dal pedicerd, *resistence design*, sono le massime sopportabili dalla sezione per un determinato tipo di sollecitazione esterna agente, mentre quella contraddistinta dal pedice brd è quella limite di stabilità per carico di punta. Il calcolo dei parametri della sollecitazione rd e brd segue le indicazioni fornite dalla normativa Euro-codice EN 1993-1-1:2005, in particolare:

$$N_{rd} = A \cdot fy \quad (4.4)$$

$$N_{brd} = \chi \cdot N_{rd} \quad (4.5)$$

$$My_{rd} = Wy \cdot fy \quad (4.6)$$

$$Mz_{rd} = Wz \cdot fy \quad (4.7)$$

$$Mt_{rd} = Wt \cdot \frac{fy}{\sqrt{3}} \quad (4.8)$$

$$Vy_{rd} = Ay \cdot \frac{fy}{\sqrt{3}} \quad (4.9)$$

$$Vz_{rd} = Az \cdot \frac{fy}{\sqrt{3}} \quad (4.10)$$

Dove:

fy: tensione di snervamento del materiale [kN/cm^2]

A: area della sezione [cm^2]

χ: fattore per l'instabilità a carico di punta degli elementi compressi.

Wy: modulo di resistenza a flessione rispetto all'asse y [cm^2]

Wz: modulo di resistenza a flessione rispetto all'asse z [cm^2]

Wt: modulo di resistenza a torsione attorno all'asse x [cm^2]

Ay: area di taglio della sezione se sollecitata lungo all'asse y nel riferimento relativo alla sezione della trave [cm^2]

Az: area di taglio della sezione se sollecitata lungo all'asse z nel riferimento relativo alla sezione della trave [cm^2]

Occorre sottolineare che Wy, Wz, Wt, Ay, Az sono proprietà geometriche della sezione e che il fattore χ, relativo al caso di compressione considerando l'instabilità a carico di punta, è calcolato servendosi dell'espressione seguente:

$$\chi = \frac{1}{\Phi + \sqrt{\Phi^2 - \bar{\lambda}^2}} \leq 1 \quad (4.11)$$

$$\bar{\lambda} = \sqrt{\frac{A \cdot fy}{N_{cr}}} \quad (4.12)$$

$$\Phi = 0.5 \cdot \left[1 + \alpha \cdot \left(\bar{\lambda} - 0.2\right) + \bar{\lambda}^2\right] \quad (4.13)$$

Dove:

α: *imperfectionfactor* è un valore tabulato e dipendente dal tipo di sezione considerata.

N_{cr} è il carico di punta critico espresso dalla seguente formula:

$$N_{cr} = \frac{\pi \cdot E \cdot I}{l_k^{\,2}} \quad (4.14)$$

Dove:

l_k: lunghezza libera di inflessione della trave [m]

E: modulo di Young del materiale

I: Momento di inerzia della sezione

Nel caso in esame l_k corrisponde alla distanza tra due nodi in cui converge più di un elemento. Dato che il sistema di riferimento relativo alla sezione considera sempre gli assi di riferimento principali y e z, anche i carichi agenti sulla sezione vengono scomposti lungo tali direzioni. Di conseguenza anche il carico critico deve essere determinato lungo tali direzioni e quindi, considerando il momento d'inerzia Iyy, si giunge a N_{cry}, mentre utilizzando Izz, a N_{crz}. Ciò porta a determinare i quattro parametri$\bar{\lambda}_y, \bar{\lambda}_z, \Phi_y, \Phi_z$ e al fine di operare sempre in sicurezza, a impiegare i valori maggiori di λ e Φ per il calcolo χ.

Osservando la formulazione delle equazioni (4.1) e (4.2) si intuisce come rappresentino una sorta di sovrapposizione degli effetti causati dalle sollecitazioni che generano sforzi normali a cui è sommato l'effetto dovuto alla torsione. La (4.3) invece è la combinazione di effetti generati da azioni che producono sforzi tangenziali. Il fatto di assegnare il maggiore tra gli indici fb ed fs come valore di utilizzazione dell'elemento significa di fatto separare gli effetti da sforzo normale più torsione da quelli di tipo tangenziale e quindi attuare una forte semplificazione. La semplificazione appare sensata considerando un elemento snello sottoposto a flessione, taglio e torsione, infatti in tale condizione il massimo dello sforzo normale si verifica in un punto dove c'è il minimo sforzo tangenziale e viceversa. Nel caso in cui si consideri anche l'azione assiale, che genera uno sforzo normale circa costante sulla sezione, sarebbe non più giustificabile la semplificazione e prendere il maggiore tra fb ed fs significherebbe operare in difetto di sicurezza, dato che in fs si è tralasciato di conteggiare anche l'effetto da sforzo assiale, che è di fatto presente.

L' *output* del blocco "Utilization" è costituito dalla lista Ut contenente i valori di utilizzazione di ciascun elemento della struttura calcolato nella maniera illustrata.

4.1.6. Ottimizzazione.

Il blocco riceve sia input che definiscono le impostazioni da assegnare all'ottimizzazione per il calocolo dei nuovi raggi ottimizzati, evidenziati in lilla in figura, che altri parametri derivati da blocchi che collaborano con esso. I primi sono Rmax, Rmin, UtiT, già descritti in precedenza, scelta, attraverso cui è impostata la funzione obbiettivo e di cui si discuterà in seguito e obj, il valore di soglia per la funzione obbiettivo. Inoltre Toggle e nitb definiscono il funzionamento dell'algoritmo con numero di iterazioni imposto, contenuto nella variabile in nitb, se la l'imput Toggle è True. Alti input derivati da blocchi precedenti sono I, che funge da contatore per le iterazioni svolte, Rad, lista dei valori dei raggi attuali delle beam e Ut, lista contenente i valori dell'utilizzazione degli elementi beam.

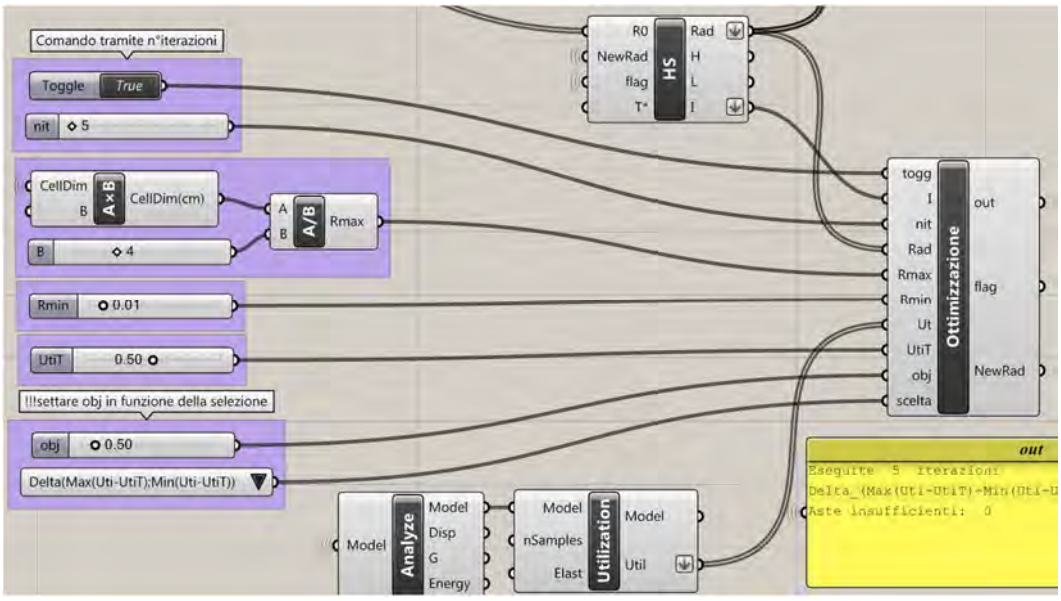

Figura 4.11:blocco Ottimizzazione

Il blocco "Ottimizzazione" produce due uscite: la lista di valori NewRad e la variabile flag. La prima rappresenta una lista di valori dei nuovi raggi calcolati dalla funzione di ottimizzazione, messi a disposizione per la creazione di un nuovo modello geometrico di struttura ottimizzata. La seconda, flag è invece una variabile di controllo che determina se sia necessario o meno eseguire una nuova iterazione.

4.1.6.1. Creazione della lista NewRad

L'operazione di ottimizzazione mira a minimizzare la densità relativa del materiale cellulare, rispettando al contempo determinati requisiti di resistenza meccanica. La diminuzione di densità relativa di un materiale cellulare è analizzata in [5]. L'autore evidenzia come la resistenza cambi drasticamente se la riduzione di densità relativa avviene per l'eliminazione di elementi costituenti la cella elementare poco utilizzati o attraverso la riduzione dell'area delle sezioni degli elementi senza però la possibilità di eliminare quelli scarsamente utilizzati. Nel primo caso si produce una riduzione del modulo elastico relativo e della σ_{sn} relativa molto più marcata che nel secondo. Tale conclusione ha portato alla scelta di ottimizzare le sezioni degli elementi senza eliminare gli elementi scarsamente utilizzati.
Gli elementi della struttura hanno sezione circolare piena, quindi la sola conoscenza del raggio è sufficiente per descrivere in modo esaustivo la sezione dell'elemento. Per il singolo elemento l'ottimizzare si configura come la successione di due operazioni fondamentali: la determinazione del nuovo raggio Rni e la sua verifica dimensionale.

a) Determinazione del nuovo raggio Rni.

   Per ogni elemento le informazioni necessarie sono raccolte nelle liste Rad e Ut, presenti come input per il blocco "Ottimizzazione". La determinazione del nuovo raggio della sezione Rni è ottenuta moltiplicando il raggio attuale R per il parametro Ratio

$$Rni = Ratio * R \quad (4.15)$$

   Riferendosi ad una delle possibili funzioni di ottimizzazione studiate, presentate al paragrafo 5.1, ad esempio la radice quadrata, il parametro Ratio assume la seguente definizione:

$$Ratio = \sqrt{\frac{Uti}{UtiT}} \quad (4.16)$$

Il fattore è rappresentato, in questo caso, dalla radice quadrata del rapporto dell'utilizzazione dell'elemento Uti rispetto all'utilizzazione target UtiT. Il rapporto tra le utilizzazioni permette di ottenere un il fattore di correzione maggiore di uno se l'utilizzazione degli elementi è superiore al target UtiT, poiché occorre far calare il livello di utilizzazione dell'elemento incrementando il suo diametro e minore di uno in caso contrario, quando occorre che il livello di utilizzazione dell'elemento aumenti per avvicinarsi al target e di conseguenza che il suo raggio diminuisca.

b) Verifica dimensionale di Rni.

   Il valore del nuovo raggio Rni va verificato per accertarsi che non generi una sezione con diametro inferiore alla minima realizzabile dal sistema di prototipazione o troppo grande se confrontata con la dimensione della cella elementare. Indicando con Rmin il minimo raggio della sezione realizzabile e con Rmax il massimo raggio compatibile con le dimensioni della cella elementare, deve sempre accadere che $Rmin \leq Rni \leq Rmax$.

   Nel caso in cu iRni oltrepassasse uno di questi due limiti, viene posto pari Rmin o Rmax in base al limite superato. .

Ripetendo i passaggi a e b descritti su ogni elemento della struttura, alla fine del ciclo di ottimizzazione la lista NewRad conterrà i valori rei raggi ottimizzati.

4.1.6.2. Funzioni obbiettivo.

La funzione obbiettivo indica il grado di ottimizzazione della struttura attuale, che presenta i raggi indicati come R al paragrafo precedente. A rigore di logica la determinazione della funzione obbiettivo dovrebbe essere svolta prima del calcolo dei nuovi raggi Rni, ma essendo entrambe le operazioni contenute nel blocco "Ottimizzazione" non c'è differenza nell'eseguire prima l'una o l'altra.

Per quanto concerne la funzione obbiettivo, tramite la variabile chiamata scelta, è possibile scegliere tra quattro modalità di computo descritte. La funzione obbiettivo è valutata considerando gli elementi con diametro maggiore di $2R_{min.}$ Di seguito vengono descritte le quattro funzioni obbiettivo selezionabili.

Massimo scostamento da UtiT.

Indicata come MaxSco, la *function* individua il massimo scostamento percentuale, superiore o inferiore, dell'utilizzazione degli elementi rispetto al target UtiT. Lo scostamento inferiore è calcolato sugli elementi con Uti<UtiT , viene chiamato Δinf, lo scostamento superiore è calcolato invece sugli elementi con Uti>UtBTe viene indicato Δsup. Valgono pertanto le seguenti espressioni:

$$\Delta inf = UtiT - Uti \quad (4.17)$$

$$\Delta sup = Uti - UtiT \quad (4.18)$$

La funzione obbiettivo è la seguente, calcolata su gli elementi che compongono la struttura.

$$Fobj = \frac{Max(\Delta_{sup}; \Delta_{inf})_i}{UtiT} * 100 \quad (4.20)$$

Massima variazione relativa del raggio tra due iterazioni.

Indicata come Max|DR|, la *function* individua il massimo valore della variazione relativa del raggio di un elemento tra due iterazioni successive. La variazione relativa del raggio di un elemento tra due iterazioni successive è indicata come DR e vale:

$$DR = \frac{|Rni - R|}{R} * 100 \quad (4.21)$$

Considerando gli elementi i che compongono la struttura, la funzione obbiettivo è la seguente:

$$Fobj = Max(DR)_i \quad (4.22)$$

Distanza tra il massimo scostamento superiore da UtiT e minimo scostamento inferiore da UtiT.

La *function* è indicata come DistSco, considerando le definizioni date di Δsup e Δinf, determina il massimo scostamento superiore Max(Δsup) e il massimo scostamento inferiore Max(Δinf), relativi a tutti gli elementi della struttura. La funzione obbiettivo è data dalla distanza tra i due:

$$Fobj = \frac{Max(\Delta sup) - Max(\Delta inf)}{UtiT} * 100 \quad (4.23)$$

Sommatoria della variazione relativa dei raggio tra due iterazioni successive

La *function* è indicata come Sum|DR|, considerando le definizione data di DR senza eseguire però la moltiplicazione per rendere il valore percentuale, determina la funzione obbiettivo nel modo seguente:

$$Fobj = \sum_i DR_i \quad (4.24)$$

4.1.6.3. Verifica Fobj.

La verifica della funzione obbiettivo serve per stabilire se la struttura analizzata soddisfi i requisiti di ottimizzazione oppure no. Questa è rappresentare dalla seguente disequazione:

$$Fobj \leq obj \quad (4.25)$$

La verifica della funzione obbiettivo a livello dello schema di figura 4.11 si traduce nel determinare il valore della variabile di controllo flag

Condizione di terminazione

La variabile di controllo flag, che determina l'esecuzione di un nuovo passo di iterazione a seconda che sia imposta su *False* o su *True*, dipende dal raggiungimento o meno della condizione di terminazione. La verifica positiva sulla funzione obbiettivo implica che la condizione di terminazione del ciclo è stata raggiunta e perciò la variabile di controllo flag è posta su *False*. Anche la comparsa di elementi insufficienti prima che la verifica sulla funzione obbiettivo sia soddisfatta implica il verificarsi della condizione di terminazione. Infatti dopo la comparsa degli elementi insufficienti, l'algoritmo non genera configurazioni strutturali che non tendono a convergere verso soluzioni con livelli di utilizzazione prossimi a UtiT.

4.1.7. Implementazione di cicli iterativi.

Il componente "Hoopsnake", detto "HS", in collaborazione con il componente "Ottimizzazione" che lo comanda tramite la variabile di controllo falg, implementa cicli iterativi con lo scopo di permettere ai dati trattati di interagire con i blocchi operazionali dell'ambiente *Grasshopper*. "Hoopsnake" opera semplicemente trasferendo i dati da *input* all'*output* sotto il controllo della variabile booleana flag. Se la variabile è *True* allora il passaggio di dati è permesso, se *False* è negato.

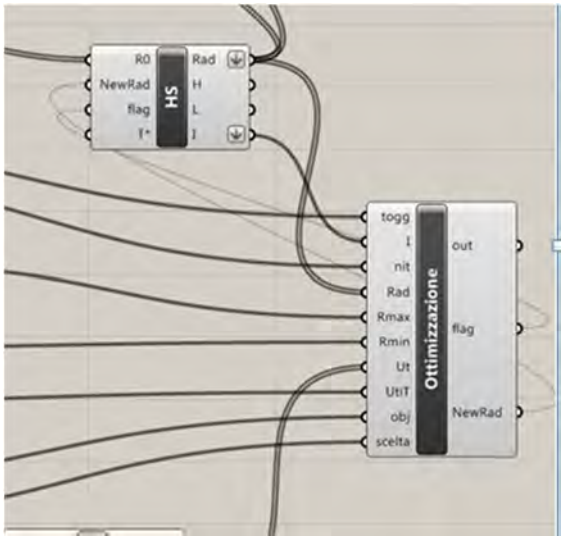

Figura 4.11: componente "Hoopsnake" per l'implementa

zione di procedure iterative.

Il componente interagisce con lo script "Ottimizzazione", infatti l'*output* Rad viene passato in*input* al blocco "Ottimizzazione" che lo modifica, creando la lista NewRad. Questa viene di nuovo fatta ritornare in *input* ad"*HS*", per poi essere passata in *output* e da qui inviata, oltre che nuovamente ad "Ottimizzazione", anche ai componenti che definiscono le dimensioni delle sezioni degli elementi *beam*. Questa configurazione permette di creare una struttura simile a quella di un ciclo iterativo for.
Al passo zero lista NewRad è vuota, menteR0 contiene la lista dei raggi iniziali degli elementi, perciò "*HS*" trasferisce R0all'output Rad che viene mandato sia verso il componente "O-Profile" che verso "Ottimizzazione", dando così inizio all'iterazione dei dati. Dopo che lo script di ottimizzazione ha prodotto la lista NewRad, "*HS*" si trova in input due liste di valori R0 e NewRad, oltre al segnale di controllo flag. Se il segnale di controllo è *True* il componente compie un nuovo passo, e trasferisce la seconda lista, NewRad, all'*output*, ignorando di fatto la prima, altrimenti, se flag è *False*, interrompe il passaggio da *input* ad *output* bloccando il ciclo.

4.1.8. Modello virtuale.

Terminata la della procedura iterativa con il raggiungimento della condizione di terminazione, il modello virtuale è ottenuto semplicemente facendo un *bake* della *mesh* posta in uscita al componente "BeamView"

4.2.Procedura iterativa di ottimizzazione delle *Beam* e delle *Shell.*

Riguarda la progettazione e ottimizzazione automatica delle strutture cellulari costituite da mesostruttura ed esostruttura tramite un algoritmo deterministico iterativo. Per i problemi legati alla comparsa di elementi insufficienti, identici a quelli riscontrati nella procedura descritta nel paragrafo 4.1, si rende necessario effettuare una ottimizzazione separata della mesostruttura e dell'esostruttura. La separazione dell'ottimizzazione non risolve di per se il problema, ma permette di by-passarlo parzialmente. Dato che si è

riscontrato come tale problema affligga soprattutto gli elementi dell'esostruttura, separando l'ottimizzazione si evita di interrompere prematuramente l'ottimizzazione della mesostruttura.

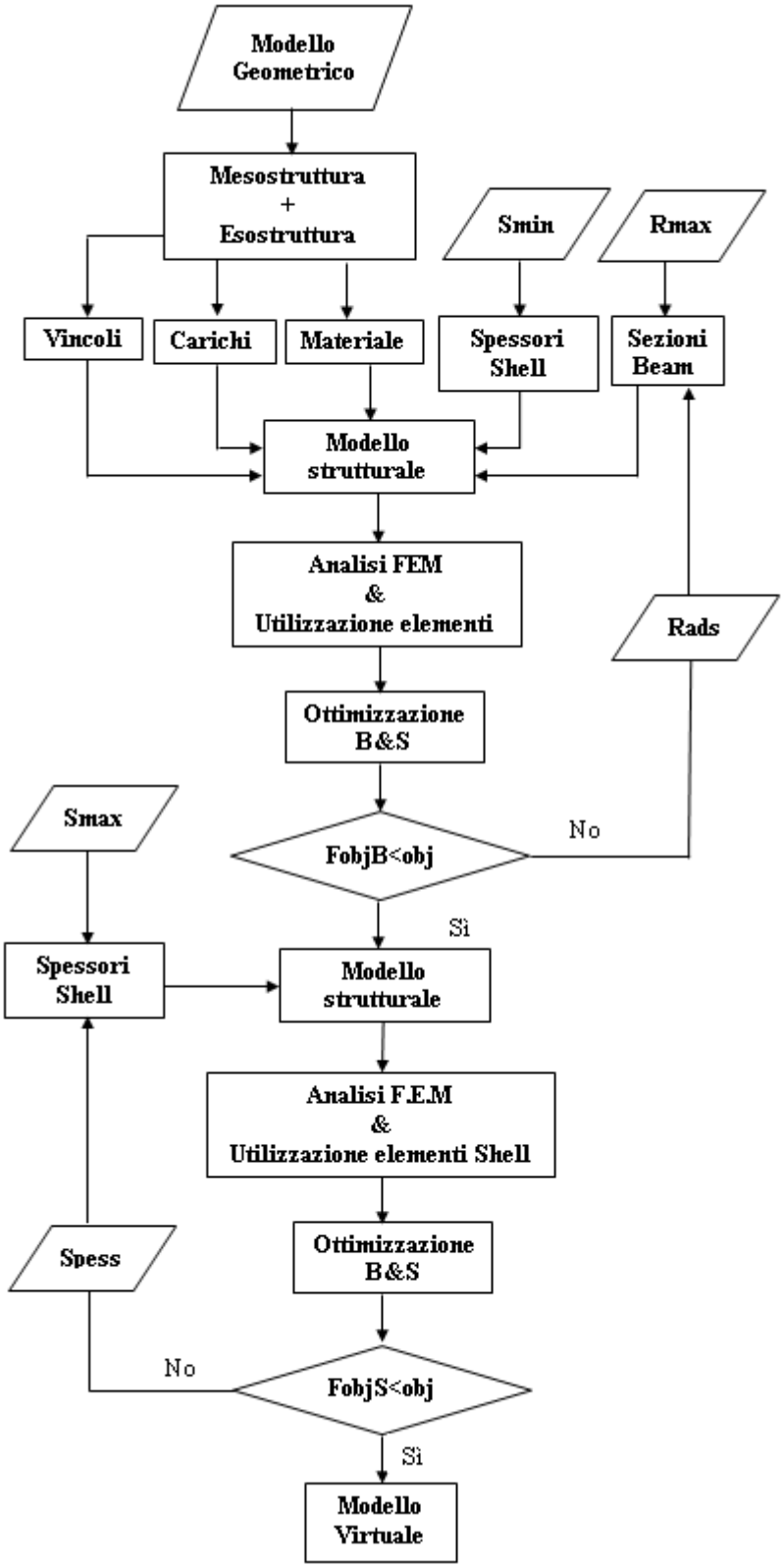

Figura 4.12: schema della procedura di ottimizzazione iterativa delle Beam e delle Shell.

Inizialmente la procedura effettua l'ottimizzazione dei soli elementi *beam*, che formano la mesostruttura, operando su una configurazione nella quale gli elementi *shell* vengono mantenuti, al passare delle iterazioni, sempre a spessore minimo. In pratica si vuole ottimizzare la mesostruttura considerando il minimo effetto dell'esostruttura. Le iterazioni sul modello descritto proseguono fino a che non viene raggiunta la condizione di terminazione analoga a quella discussa al paragrafo 4.1.7.
Determinata la configurazione della mesostruttura, questa è considerata fissa, si attua l'ottimizzazione dell'esostruttura fino a che non viene raggiunta la condizione di terminazione.
Molti dei blocchi rappresentati nello schema di figura 4.12 sono comuni a quelli dello schema di figura 4.2, perciò per maggiori dettagli si faccia riferimento al paragrafo 4.1 che descrive la procedura iterativa di ottimizzazione delle b*eam*.

4.2.1. Mesostruttura + Esostruttura.

La mesostruttura è stata ampiamente descritta al paragrafo 4.1.2. L'esostruttura è costituita dal guscio esterno dell'oggetto e di per sé ha è già ben rappresentata dalla polisuperficie di tipo NURBS che definisce il solido. Questa rappresentazione della superficie non è però utile per assegnare le proprietà di elemento strutturale alla superficie, per fare ciò occorre partire dalla rappresentazione tramite *mesh* superficiale. Questa non è altro che una rappresentazione approssimata della superficie originale tramite una superficie poligonale costituita da triangoli che condividono tra loro vertici e lati.
Requisito fondamentale per la *mesh* è di essere *watertight*, cioè impermeabile o a tenuta stagna, in modo da rappresentare la continuità della superficie. Un'ulteriore requisito fondamentale è che le intersezioni tra le linee di definizione della mesostruttura e la *mesh* corrispondano a vertici della *mesh*. Per ottenere la *mesh* coi requisiti voluti si rivela di fondamentale importanza il componente "MeshBreps" di Karamba.

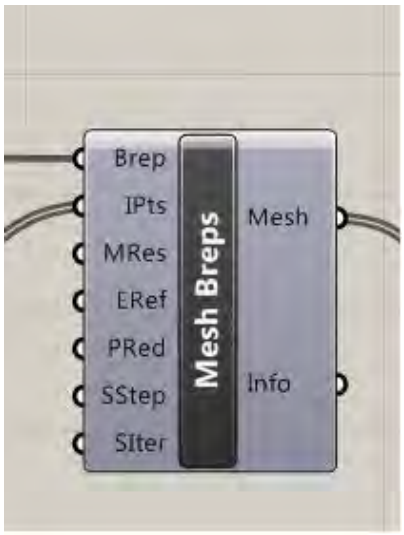

Figura 4.13: componente MeshBreps

In figura 4.13 si vede il componente con gli *input* presenti sul lato sinistro e gli *output* sul destro. L'input rappresentato dalla geometria del solido di cui generare la *mesh* superficiale è Brep, quello dei punti per cui imporre dei vertici della *mesh* è IPts, mentre gli altri ingressi definiscono gli aspetti parametrici della *mesh* da generare. In particolare MRes definisce la risoluzione della mesh intesa come lunghezza *target* per i lati dei triangoli che la compongono, ERef definisce l'infittimento della mesh sui bordi come valore da imporre in un *range* da 0 a 1. L'output è invece costituito dalla mesh di superficie.

4.2.2. Spessori shell.

La *mesh* ottenuta con il componente MeshBreps è una entità singola formata molti triangoli che non possono essere presi singolarmente. Assegnando le proprietà di *shell* agli elementi triangolari della *mesh* così ottenuta non si potrebbe effettuare un ottimizzazione sullo spessore dei singoli triangoli, ma si potrebbe solamente ottimizzare lo spessore in riferimento all'elemento più sollecitato. Per ottenete maggiore flessibilità e quindi la possibilità di assegnare gli spessori a ciascun elemento triangolare occorre poter operare su ogni triangolo della *mesh*. Per fare ciò occorre esplodere la *mesh* data

dal componente “MeshBreps”, ottenendo le singole facce triangolari e da queste ricostruire una nuova *mesh*.

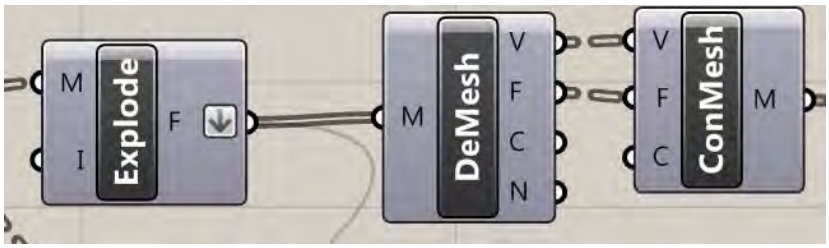

Figura 4.14: ricostruzione della mesh sulle singole facce

La *mesh* risultante non è più un’entità unica ma è formata da un numero di entità pari al numero di facce che componevano la mesh che entra in *input* al componente “Explode” di figura 4.14.

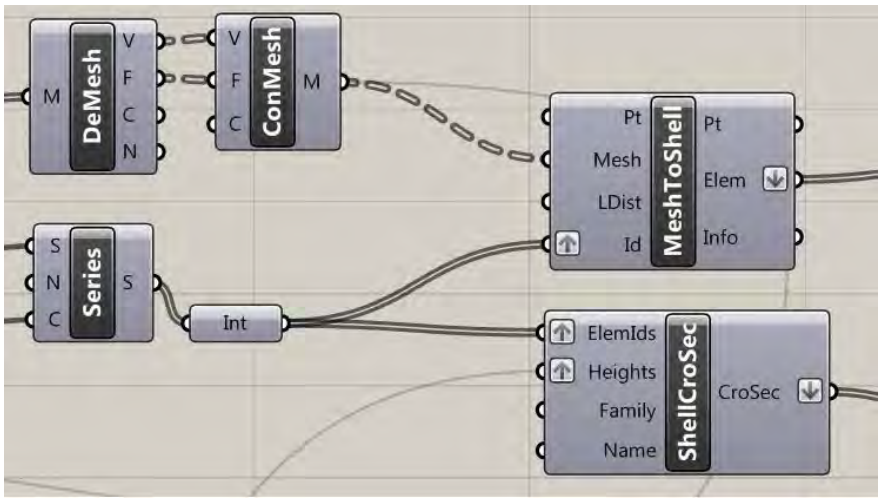

Figura 4.15: assegnazione degli spessori agli elementi shell

Grazie al componente “MeshToShell” di Karamba è possibile assegnare le proprietà di elemento strutturale *shell* alla *mesh*. Per la modalità nella quale è stata realizzata la *mesh* è possibile assegnare ad ogni faccia lo spessore utilizzando il componente “ShellCroSec” di Karamba. Questo componente agisce in maniera concettualmente identica a “O-Profile” descritto nel paragrafo 4.1.

4.2.3. Ottimizzazione B&S.

Da punto di vista delle operazioni svolte, la sezione è molto simile a "Ottimizzazione B", descritta nel paragrafo 4.1, solo che opera sia su elementi *beam* che su *shell*. Di conseguenza l'*output* NewRad, viene ora ad essere formato dalla somma delle liste Rads e Spess, contenenti rispettivamente i raggi e gli spessori ottimizzati.
Dato che si utilizzano due tipi di elementi strutturali differenti, saranno presenti due valori per la funzione obbiettivo, quella che si riferisce agli elementi *beam* e quella degli elementi *shell*, FobjB e FobjS. La determinazione di FobjB è identica a quella presentata nel paragrafo 4.1.6, mentre quella di FobjS è sempre concettualmente identica, ma svolta sugli elementi *shell*

4.2.4. Verifica FobjB.

Dato che la lista Rad contiene sia i valori dei raggi delle *beam*, che gli spessori delle *shell*, per separarli si considerano le sotto-liste Rasds e Spess ottenute dividendo la lista originale in due parti in corrispondenza della posizione data dall'indice Lng. Questo valore corrisponde al numero di elementi *beam* presenti nel modello.
Per quanto riguarda la verifica degli elementi beam è identica a quella descritta nel paragrafo 4.1.7.

Implementazione del ciclo iterativo di ottimizzazione mesostrutturale

Il ciclo iterativo che riguarda l'ottimizzazione della mesostruttura è implementato alla stessa maniera descritta nel paragrafo 4.1.7, con la differenza che ora occorre impostare anche il numero massimo di iterazioni possibili, tramite il parametro nitb.
Altra peculiarità sta nel fatto

4.2.5. Verifica FobjS

La verifica riguarda questa volta gli elementi presenti nella sotto lista Spess che individua gli elementi *shell*.

Lo script determina i valori di NVG, FobjS e Sins per ogni iterazione. FobjS è la funzione obbiettivo per gli elementi *shell*, Sins rappresenta il numero di *shell* insufficienti, di definizione analoga a quella di *beam* insufficiente, mentre NVG è il numero di *beam* non verificate. La condizione di terminazione riguarda, oltre al verificarsi della disequazione FobjS≤obj, anche la comparsa di elementi NVG oppure quella di Sins. La comparsa di elementi NVG è da monitorare in quanto, essendo la mesostruttura non più modificabile, una volta che un elemento diventa non verificato non è più possibile agire direttamente per far rientrare il valore di utilizzazione sotto la soglia di sicurezza fissata da UtBT.

Implementazione del ciclo iterativo di ottimizzazione esostrutturale.

Per la gestione del secondo ciclo di ottimizzazione rappresentato nello schema 4.14, che si riferisce all'ottimizzazione dell'esostruttura, si opera come illustrato di seguito.

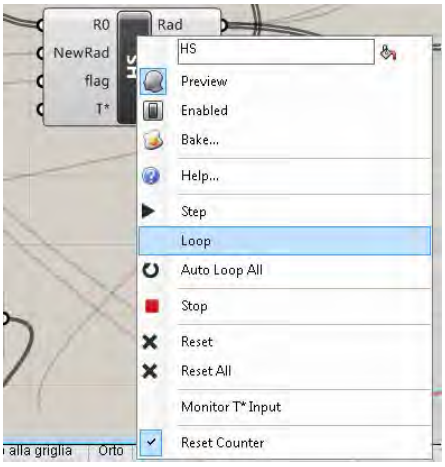

Figura 4.16: comando loop di HS

Per prima cosa occorre impostare il parametro nitb, *input* dello script ottimizzazione, con il numero di iterazioni svolte dal primo ciclo iterativo. Fatto ciò basta solamente aprire il menù a tendina mostrato in figura 4.16 e cliccare step per eseguire una iterazione. Purtroppo questo secondo ciclo non è automatico, ma occorre che sia implementato manualmente e controllato dall'operatore al fine di individuare la

condizione di terminazione. Per far ciò risulta fondamentale la stampa a video dei valori impiegati nella verifica sulla funzione obbiettivo, NVG, FobjS e Sins.

4.3.Procedura di ottimizzazione basata sull'algoritmo genetico.

La procedura non è, come le precedenti, di tipo deterministico, ma di tipo stocastico e basata sull'impiego di un algoritmo genetico per la ricerca di soluzioni ottimali.

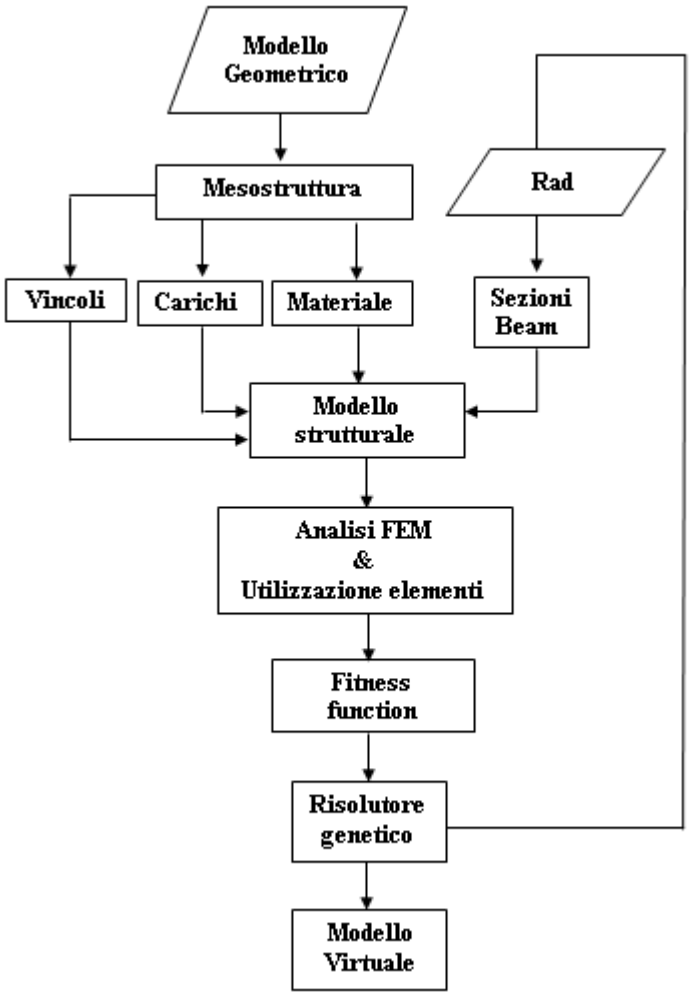

Figura 4.17: schema della procedura di ottimizzazione basata sull'algoritmo genetico G

Dallo schema si osserva che la struttura di controllo di tipo iterativo è stata sostituita dal **Risolutore genetico**, che interagisce coi blocchi **Fitness function** e **Rad**.

4.3.1. Risolutore genetico.

Il risolutore genetico Galapagos esplora lo spazio delle possibili soluzioni del problema facendo evolvere, in maniera iterativa, una popolazione costituita da un certo numero di soluzioni verso quella ottimale

Si consideri inizialmente una popolazione di soluzioni costituita in modo casuale e che copra il più possibile lo spazio delle soluzioni a disposizione. Ogni soluzione è una n-upla, chiamata Rad, contenente i valori dei raggi che definiscono le sezioni degli elementi del modello strutturale. Rad, in collaborazione coi blocchi presentati in figura 4.17, determina una configurazione strutturale le cui prestazioni vengono stabilite effettuando innanzitutto un'analisi FEM, a cui segue il calcolo dei livelli di utilizzazione degli elementi *beam* e la valutazione della fitness. Effettuando tali operazioni per tutte le soluzioni della popolazione e classificandole in ordine di fitness, crescente o decrescente, in base al dover ricercare le soluzioni che massimizzano o minimizzano la *fitness function*, si è compiuto il primo passo della procedura.

Al passo successivo la popolazione viene sostituita con una nuova generazione di soluzioni formata a partire dalle soluzioni precedenti e impiegando una certo numero di operatori genetici, mutuati dalla genetica naturale, descritti al capitolo 3.2. Sulla nuova popolazione vengono ripetuti i passi descritti in precedenza fino al completamento dell'iterazione 2 e cosi di seguito fino a quando non vengono raggiunte le condizioni di terminazione.

Per quanto concerne la condizione di terminazione, è possibile stabilirla in diversi modi e ognuno può agire simultaneamente interrompendo l'esecuzione non appena si sia verificata una delle condizioni. Al capitolo 3.2.2 è descritta la finestra di dialogo di Galapagos Editor, all'intero della barra degli strumenti Options è possibile stabilire le condizioni di terminazione, che sono sia la durata temporale delle iterazioni, che il valore soglia sotto il quale interrompere le iterazioni qualora la migliore soluzione presenti valore di fitness minore, viceversa nel caso di massimizzazione e il massimo numero di iterazioni effettuate senza ottenere miglioramenti nella soluzione a migliore fitness.

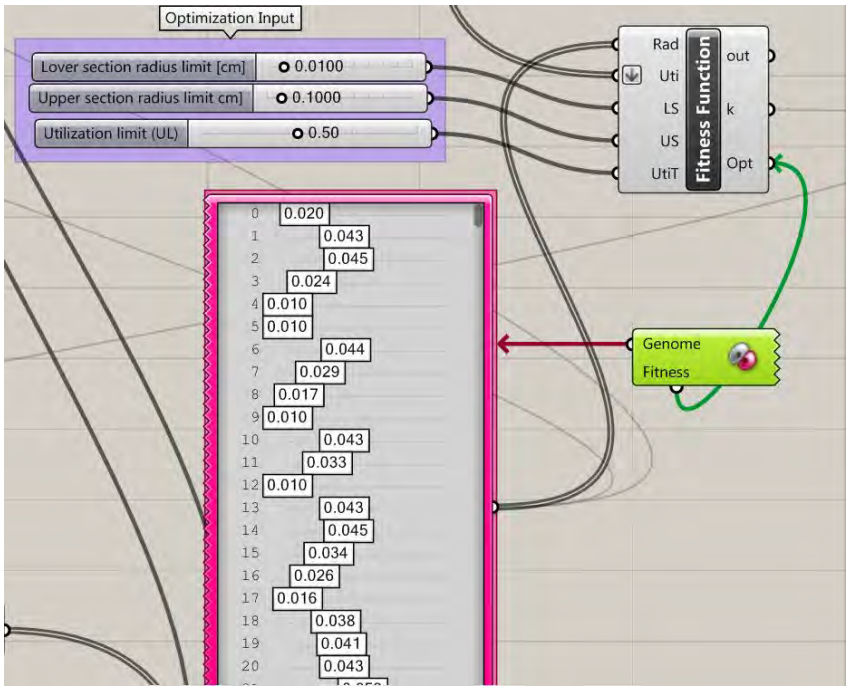

Figura 4.18: interazione tra Risolutore genetico, Fitness function e Gene pool

In figura 4.18 è rappresentata l'interazione del risolutore genetico, evidenziato in verde, con i componenti "FitnessFunction" e "GenePool". Il primo si occupa di calcolare il valore delle funzione di fitness per ogni soluzione e verrà descritto in seguito, mentre il secondo è un contenitore che mostra i valori assunti dai parametri che definiscono la soluzione in esame. Da quest'ultimo parte in uscita la lista Rad che, come illustrato in figura 4.17, è necessaria per definire le sezioni degli elementi *beam*. A differenza delle procedure B e B&S, il valore di Rad non è impostabile dall'utente nemmeno come singola soluzione dell'iterazione iniziale.

4.3.2. Fitness function.

Il valore della funzione di fitness è una misura della adeguatezza della soluzione esaminata. Gli input del componente sono le liste Rad e Uti, contenenti i valori dei raggi e delle utilizzazioni degli elementi, oltre all'utilizzazione target UtiT. Opt è invece l'output corrispondente al valore della funzione obbiettivo per la soluzione esaminata
Per la singola asta, sia op lo scarto quadratico dell'utilizzazione Uti rispetto al target UtiT, espressa dall'equazione seguente:

$$op = \left(\frac{UtiT - Uti}{UtiT}\right)^2 \quad (4.26)$$

La radice quadrata della sommatoria degli scarti quadratici op, corrispondente allo scarto quadratico medio delle utilizzazioni rispetto all'utilizzazione target, è la *Fitness function* da minimizzare Opt.

$$Opt = \sqrt{\sum op} \quad (4.27)$$

Il componente "Fitness Function" mostrato in figura 4.18 presenta anche ulteriori *input:* LS e US, che corrispondono al minimo e al massimo raggio ammissibile per le sezioni. La necessità della loro conoscenza risiede nel fatto che, per favorire la convergenza dei raggi degli elementi con utilizzazione prossima a zero verso il minimo, si è riscontrato utile implementare un criterio differente rispetto a quello rappresentato dall'equazione 4.26. Questi elementi, non essendo caricati, non mostrano variazioni di utilizzazione al variare del raggio della sezione. L'equazione (4.26) darebbe quindi risultati simili al variare dei diametri degli elementi, poiché l'utilizzazione resterebbe sempre prossima a zero. Il criterio di convergenza per elementi non caricati è il seguente:

$$op = \left(\frac{R - LS}{LS}\right)^2 \quad (4.28)$$

L'equazione (4.28) è da applicarsi agli elementi che presentano utilizzazione Uti<0.0001.

Per gli elementi che presentano utilizzazione superiore a UtiT si è anche introdotta una condizione di penalizzazione:

$$op = 10 * \left(\frac{R - LS}{LS}\right)^2 \quad (4.29)$$

In tal modo vengono penalizzati gli elementi sovrautilizzati rispetto a UtiT.

5. Casi applicativi

5.1 Esempi di applicazione della procedura iterativa di ottimizzazione beam.

Per verificare il funzionamento del metodo proposto con la procedura B si sono analizzati alcuni casi esemplificativi di struttura cellulare. Lo scopo della seguente analisi è soprattutto quello di evidenziare i problemi della procedura al fine di proporre miglioramenti.

5.1.1. Parallelepipedo 10x10x50 a cella GAM sottoposto a flessione e taglio.

Si consideri il volume di un parallelepipedo di dimensioni 10x10x50mmriempito con una struttura cellulare a cella GAM di lato 3mm. La struttura sia formata solamente dalle celle, da intendersi costituite da elementi *beam*, tralasciando quindi l'assegnazione di elementi strutturali di tipo *shell* alla superficie del parallelepipedo. Si vincolino i nodi corrispondenti ad una delle due basi quadrate, di lato 10mm e si pongano dei carichi sui nodi della base opposta in maniera che la risultante delle forze agenti sui nodi sia verticale e di intensità pari a 7N.Questa tipologia di carico è stata scelta in quanto genera una distribuzione lineare del parametro della sollecitazione momento lungo l'asse x.

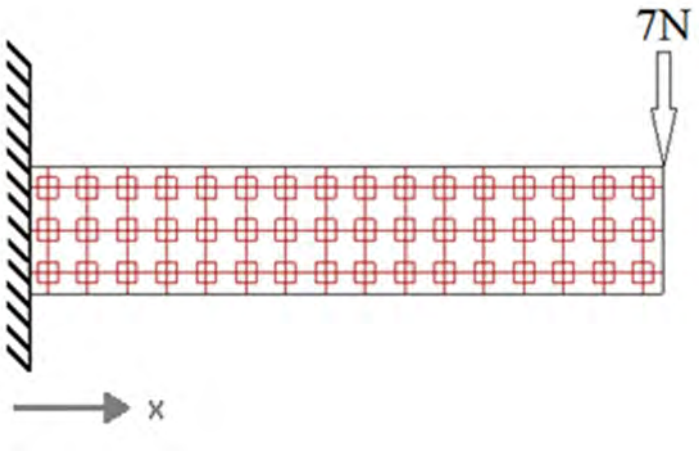

Figura 5.1: schema della struttura a cella GAM caricata e vincolata

La struttura risultante, schematizzata in figura 5.1, ha le caratteristiche riportate in tabella seguente:

parallelepipedo 10x10x50					
cella	lato[mm]	N°celle	N° beam	N°nodi	R beam[mm]
GAM	3	144	3993	2802	0.75

Tabella 5.1: caratteristiche della struttura schematizzata in figura 5.1.

Il materiale della struttura sia il poliammide PA2105 prodotto dalla EOS e avente le seguenti caratteristiche meccaniche:

PA 2105			
σsn[Mpa]	E[Gpa]	G[Gpa]	ρ[kg/m3]
54	1.85	0.748	950

Tabella 5.2: caratteristiche del materiale PA 2105.

Al modello si applichi l'algoritmo di ottimizzazione B, impostando R_{min} e R_{max} rispettivamente a 0,2 e 0,75mm e utilizzando come funzione di ottimizzazione la funzione radice quadrata, equazione (4.16) riportata di seguito:

$$Rni = R\sqrt{\frac{Ut[i]}{UtiT}}$$

Per quanto riguarda la funzione obbiettivo è stato posto il massimo scostamento dal target UtiT, per maggiori dettagli si rimanda al paragrafo 4.1.6.1.

Dallo schema di figura 4.2 si desume che la struttura di partenza ha tutte le *beam* a raggio R_{max}, quindi quella con la maggiore resistenza realizzabile.

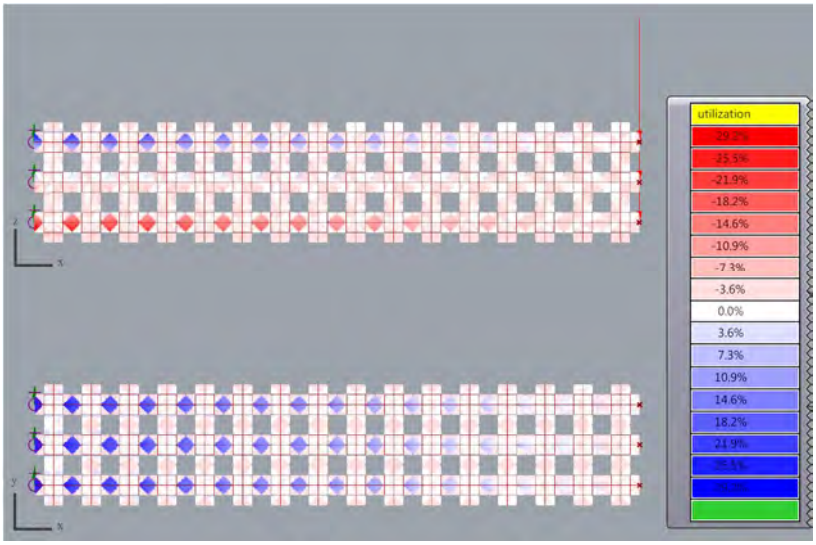

Figura 5.2: livelli di utilizzazione degli elementi per la struttura iniziale

La figura 5.2 mostra l'utilizzazione degli elementi beam: si osserva come le maggiori utilizzazioni si abbiano vicino alla base vincolata, dove il momento flettente è massimo e come si riducano allontanandosi dalla base seguendo la direzione x. Dalla vista x-z si nota inoltre che gli elementi posti sull'asse neutro della sezione del parallelepipedo sono a loro volta poco utilizzati. Dato che la massima utilizzazione registrata è circa 30%, significa che nessun elemento è sovraccaricato, ma è anche indice di come la struttura sia ottimizzabile.

Nella tabella seguente sono riportati gli indici prestazionali della struttura relativi alle prime cento configurazioni strutturali individuate dall'algoritmo. Si ricorda il significato dei termini: NVG indica il numero di elementi che non sono verificati a causa di

utilizzazione superiore al 10% del target, per Uti>55%, ins conteggia il numero di elementi che sono sia nella condizione NVG che al diametro massimo.

iter.	NVG	Utimax%	Utimin%	Ins	Fobj%
0	0	30	8,9	0	98,2
1	2515	323,6	18,5	0	547,3
2	0	51,4	16	0	68,1
3	1479	87,1	33,2	0	74
4	4	55,8	26,4	4	47,1
5	154	58,4	26,3	4	47,5
6	4	57,3	29,5	4	40,9
7	4	57,8	32,8	4	34,5
8	4	58,1	29,6	4	40,9
9	4	58,5	29,1	4	41,9
10	4	58,7	36,2	4	27,5
11	4	58,9	40,2	4	19,4
12	4	59	39	4	22,7
13	4	59,2	37,6	4	24,8
14	4	59,4	37,2	4	25,6
15	4	59,4	39,7	4	20,69
20	4	59,9	38,4	4	23,2
30	4	60,7	42,8	4	21,4
40	4	61,2	43,3	4	22,4
50	4	61,7	39,9	4	23,4
100	5	64,1	36,2	5	28,2

Tabella 5.3: prospetto con le iterazioni effettuate dalla procedura B

Dal prospetto si osserva inizialmente un andamento poco regolare degli indici NVG e Utimax %. Infatti dalla prima alla seconda iterazione passano da valori elevati, 2515 e 323,6%, a valori molto più bassi, 0 e 51,4%. Successivamente si nota invece un andamento più regolare.

Merita di essere evidenziatala presenza di elementi insufficienti a partire dalla quarta iterazione. Il loro numero resta costante al passare delle iterazioni ,ma in contemporanea si verifica una costante crescita dell'utilizzazione massima, che passa da 55,8% della quarta iterazione a 64,1 della centesima iterazione. La cosa deve far preoccupare in quanto indica che, una volta raggiunta la condizione di insufficienza per un elemento,

l’algoritmo non riesce a porre rimedio alla continua crescita della sua utilizzazione. Infatti si verifica che gli elementi che raggiungono la massima utilizzazione sono anche quelli con raggio massimo

La spiegazione di tale fenomeno va ricercata nel fatto che l’unico modo che ha l’algoritmo per opporsi all’utilizzazione crescente di un suo elemento è di aumentarne il diametro, eventualmente fino al massimo possibile. Dopo di che non esistono ulteriori possibilità per l’elemento di far fronte ad aumenti di carico. A tale considerazione ne va aggiunta un'altra: l’algoritmo B tenta di far raggiungere l’utilizzazione obbiettivo agli elementi poco caricati assottigliandoli. Questo provoca una diminuzione della rigidezza di tali elementi e un conseguente trasferimento di carico verso gli elementi più rigidi della struttura, a diametro maggiore. Considerando un elemento sollecitato che condivide almeno un nodo con altri elementi poco sollecitati, nel caso in cui questo diventasse insufficiente il meccanismo del trasferimento di carico non farebbe altro che peggiorare la situazione dell’elemento insufficiente andando a caricarlo ancora di più e quindi facendo crescere la sua utilizzazione. Perciò alla comparsa di elementi insufficienti la struttura non può più tendere verso un miglioramento, ma solo peggiorare le proprie prestazioni. Da tali considerazioni si capisce come, in caso di comparsa di elementi insufficienti, occorra interrompere la procedura di ottimizzazione.

Nel caso esaminato, la procedura di ottimizzazione si sarebbe dovuta fermare alla quarta iterazione con la comparsa dell’indice ins diverso da zero. In tal caso la utilizzazione minima degli elementi considerati, quelli non a diametro minimo, sarebbe pari a 26.3%.

Funzioni di ottimizzazione

Per incrementare l’utilizzazione degli elementi, l’algoritmo agisce sfinandoli: appare chiaro come sia cruciale il comportamento della funzione di ottimizzazione nella zona di ascisse da 0 a 1. La funzione di ottimizzazione implementata appare troppo brusca a causa della forte alternanza di risultati mostrati e per il fatto che già all’iterazione quattro mostra la comparsa di elementi insufficienti.

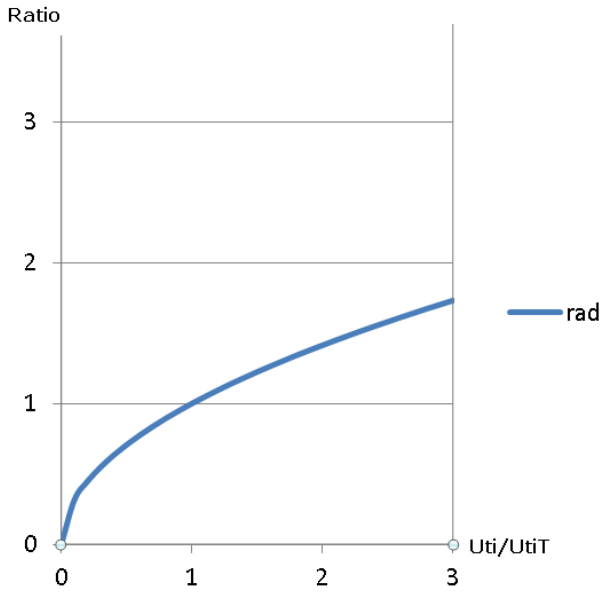

Figura5.3: andamento di rad in funzione di Uti/UtiT

Considerando il grafico della radice quadrata del parametro Uti/UtiT, si osserva che nella zona delle ascisse minori di 0,5 la funzione assume valori molto piccoli, ciò significa una repentina riduzione del valore di diametro degli elementi poco caricati.
Una alternativa potrebbe essere data da una funzione con le seguenti caratteristiche:
-per Uti/UtiT=0 abbia valore diverso da zero
-nella zona attorno all' ascissa 1 abbia pendenza bassa
- nella zona di Uti/UtiT>1 sia rapida nell'aumento di raggio
Si propone la finzione composta di equazione seguente, il cui grafico è riportato in figura 5.4.

$$Ratio = 0.5 + 0.5\sqrt{1 - \left(\frac{Uti - UtiT}{UtiT}\right)^2}; \quad \frac{Uti}{UtiT} < 1$$

$$Ratio = 4\left(\frac{Uti}{UtiT}\right)^2 - 8\left(\frac{Uti}{UtiT}\right) + 5; \quad \frac{Uti}{UtiT} > 1 \quad (5.1)$$

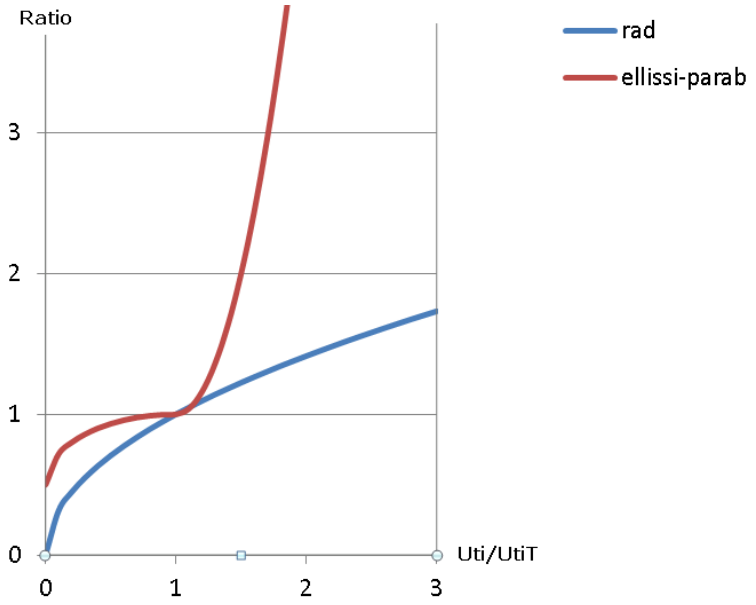

Figura 5.4: confronto tra l'andamento di rad e la funzione composta (5.1)

Nella tabella 5.4 si confrontano gli indici prestazionali delle funzioni di ottimizzazione proposte. Gli indici si riferiscono alla condizione di arresto raggiunta, riportata come intestazione della tabella 5.2

	Condizione di arresto: Fobj<10% o ins>0					
	Iter	NVG	Ins	Utimax	Utimin	Fobj%
radq	4	4	4	55,8	26,4	47,2
ellissi-parabola	32	2	2	55,0	33,3	33,4
ellissi-lineare	29	2	2	55,0	32,4	35,2

Tabella 5.4: confronto tra le funzioni di ottimizzazione proposte effettuato nella condizione di arresto della procedura

Si osserva che il numero di iterazioni impiegate dall'algoritmo per raggiungere la condizione di arresto è molto aumentato con l'impiego della funzione ellittica nel campo Uti/UtiT<1. Il vantaggio nell'impiego ellissi, piuttosto che la radice quadrata si osserva con un aumento della utilizzazione minima, che passa da 26.4 a 33.3 nel caso ellissi-parabola. Inoltre, dato che tra il caso ellissi-parabola e il caso ellissi-lineare si nota poca differenza, si intuisce come la parte più importante della funzione di

ottimizzazione sia quella nel campo 0-1. Appare opportuno sottolineare che quanto detto ha valenza per il caso specifico, in generale occorre, di volta in volta provare quale funzione si adatta meglio alle condizioni generali di vincolo e carico.

5.1.2. Parallelepipedo 10x10x50 a cella BCC sottoposto a flessione e taglio.

Si consideri un caso simile a quello descritto al paragrafo 1, con l'unica variante della cella BCC invece della GAM. Si è scelto questo tipo di cella in quanto ha un comportamento sostanzialmente differente dal punto di vista meccanico rispetto alla cella GAM. Infatti quest'ultima è una struttura in cui gli elementi lavorano principalmente a flessione, mentre la BCC è una struttura in cui gli elementi lavorano in maniera bilanciata a flessione e compressione.

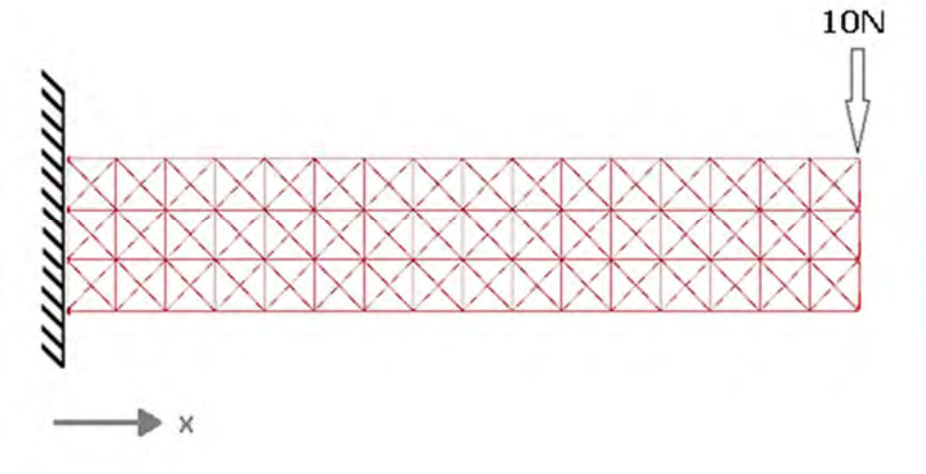

Figura 5.5: schema della struttura a cella BCC caricata e vincolata.

La struttura risultante, schematizzata in figura 5.1, ha le caratteristiche riportate in tabella seguente:

parallelepipedo 10x10x50					
cella	lato[mm]	N°celle	N° beam	N°nodi	r beam[mm]
BCC	3	144	1816	416	0.5

Tabella 5.5: caratteristiche della struttura schematizzata in figura 5.5.

Il materiale della struttura sia il poliammide PA2105 prodotto dalla EOS e avente le seguenti caratteristiche meccaniche:

PA 2105			
σsn[Mpa]	E[Gpa]	G[Gpa]	ρ[kg/m3]
54	1.85	0.748	950

Tabella 5.6: caratteristiche del materiale PA 2105.

Al modello si applichi l'algoritmo di ottimizzazione B, impostando R_{min} e R_{max} rispettivamente a 0,2 e 0,75mm, utilizzando come funzione di ottimizzazione la funzione radice quadrata e come funzione obbiettivo il massimo scostamento dal target UtiT, descritta al paragrafo 4.1.6.1.
Nella tabella seguente sono riportati gli indici prestazionali della struttura relativi alle prime cinquantanove configurazioni strutturali individuate dall'algoritmo

iter.	NVG	Utimax	Utimin	Ins	Fobj%
0	0	34,2	0	0	100
1	0	40,6	0	0	100
2	0	45,3	0,7	0	98,5
3	0	49,2	2,8	0	94,4
4	0	53	3,4	0	93
5	0	52,1	12,1	0	75,8
6	0	51,9	9,7	0	80,7
7	0	54,1	5,7	0	88,6
8	4	60,5	9,3	0	81,4
9	0	53,5	16,8	0	66,4
10	0	54	16,7	0	66,6
11	0	53,4	16,4	0	67,1
12	0	52	18,8	0	62,4
13	2	56,7	22,4	0	55,1
14	0	52,5	26,2	0	47,5
15	0	51,7	25,5	0	49
20	0	50,8	26,5	0	47
30	0	50,2	37,9	0	24,2
40	0	50,1	42,1	0	15,89

50	0	50,1	44,1	0	11,7
59	0	50,1	45,1	0	9,88

Tabella 5.7: prospetto co le iterazioni effettuate dalla procedura B.

Dal prospetto si vede che è stata individuata la condizione di arresto alla cinquantanovesima iterazione, dove la funzione obbiettivo assume il valore di 9.88%
A differenza della prova con la cella GAM, non si osserva la comparsa di elementi insufficienti a conferma di quanto affermato in precedenza, ovvero che il metodo smette di ottimizzare efficacemente la struttura alla loro comparsa. Si vede che la massima utilizzazione si stabilizza a valori molto prossimi a quella voluta, UtiT=50% gia dalla ventesima iterazione. Anche l'utilizzazione minima ha un andamento regolare e crescente dall'undicesima iterazione in poi, fino a giungere a 45.1% all'iterazione 59.
Dall'andamento dei parametri si può ipotizzare che proseguendo le iterazioni con una funzione obbiettivo più bassa, ad esempio 5%,.si dovrebbe giungere comunque a convergenza.

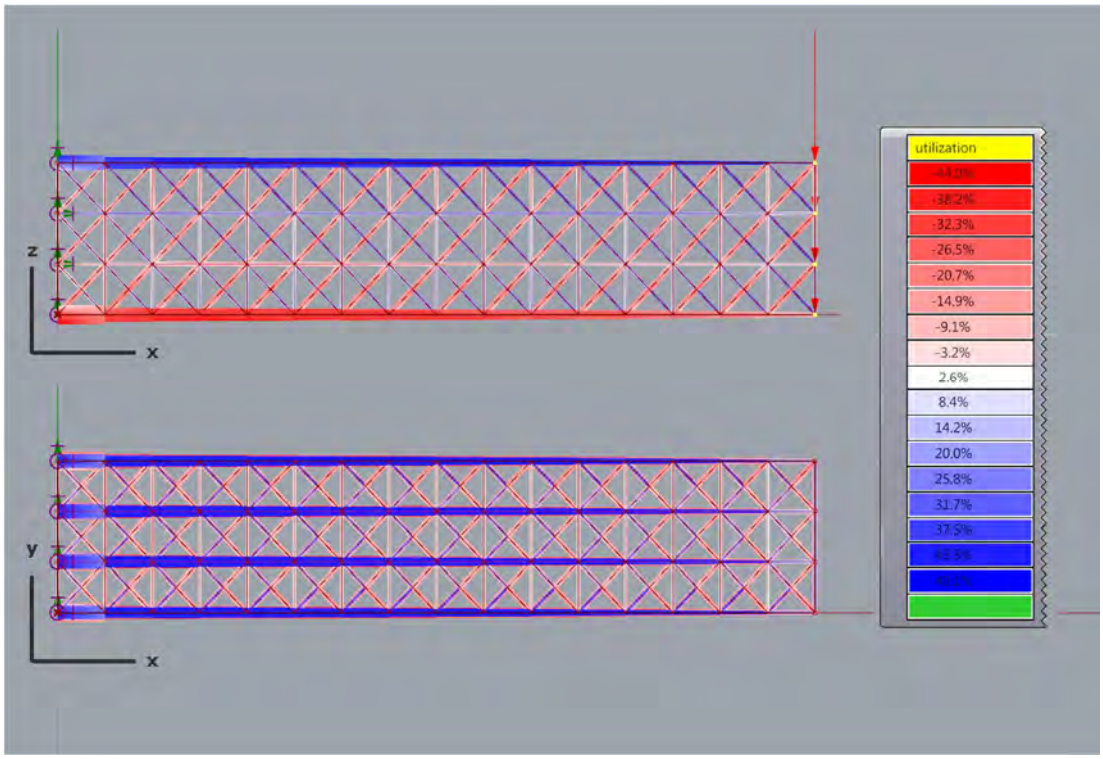

Figura 5.6: livelli di utilizzazione degli elementi per la struttura definitiva.

Il massimo diametro degli elementi a struttura ottimizzata è 0.62mm; ciò suggerisce come la struttura di partenza potrebbe anche essere ottimizzata utilizzando valori più elevati del carico.

5.1.3.Parallelepipedo 10x10x50 a cella GAM sottoposto a compressione.

Si consideri un carico di compressione di 1MPa distribuito sulla faccia quadrata di lato 10mm. La sollecitazione risultante agente sarà pertanto di intensità pari a 100N. Questa tipologia di carico è stata scelta in quanto genera una distribuzione costante del parametro della sollecitazione assiale lungo x.

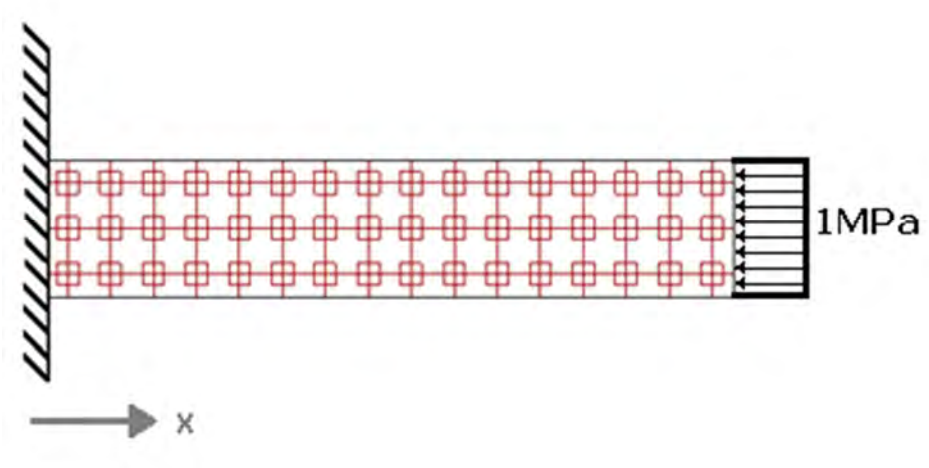

Figura 5.7: schema della struttura a cella GAM caricata e vincolata.

La struttura abbia le seguenti caratteristiche:

parallelepipedo 10x10x50					
cella	lato[mm]	N°celle	N° beam	N°nodi	R beam[mm]
GAM	3	144	3993	2802	0.5

Tabella 5.8: caratteristiche della struttura schematizzata in figura 5.7.

Il materiale della struttura sia il poliammide PA2105 prodotto dalla EOS e avente le seguenti caratteristiche meccaniche:

PA 2105			
σsn[Mpa]	E[Gpa]	G[Gpa]	ρ[kg/m3]
54	1.85	0.748	950

Tabella 5.9: caratteristiche del materiale PA 2105.

Al modello si applichi l'algoritmo di ottimizzazione B, impostando R_{min} e R_{max} rispettivamente a 0,2 e 0,75mm, utilizzando come funzione di ottimizzazione la funzione radice quadrata e come funzione obbiettivo il massimo scostamento dal target UtiT, descritta al paragrafo 4.1.6.1.
Nella tabella seguente sono riportati gli indici prestazionali della struttura relativi alle prime quaranta configurazioni strutturali individuate dall'algoritmo

iter.	NVG	Utimax	Utimin	Ins	Fobj%
0	0	33,0	0	0	100
1	0	38,1	0	0	100
2	0	41,4	7,5	0	85
3	0	43,7	9	0	82,1
4	0	45,4	9,4	0	81,2
5	0	46,7	11	0	78
10	0	49,7	35,5	0	28,9
20	0	50,1	40,8	0	18,5
30	0	50,1	44,1	0	11,8
40	0	50,0	45,1	0	9,8

Tabella 5.10: prospetto con le iterazioni effettuate dalla procedura B.

A differenza del caso sottoposto a flessione, per il carico di compressione l'algoritmo riesce a terminare senza che compaiano elementi insufficienti. Confrontando i dati di tabella 5.3 con quelli si tabella 5.10 si osserva come all'iterazione zero entrambi abbiano utilizzazione 30% circa, ma nel primo caso l'algoritmo non ha trovato una condizione di terminazione prima dell'insorgere di elementi insufficienti, mentre nel secondo caso si. Tale diversità è da attribuite alla differente natura del carico agente, nel senso che il carico flessionale genera una distribuzione del parametro della

sollecitazione lineare lungo x, mentre il carico di compressione genera una distribuzione costante.

Nella figura seguente è mostrata la struttura fornita all'iterazione quaranta, si osserva la presenza di tre colonne di elementi ingrossati che supportano il carico e il la simmetria tra la vista x-y e x-z.

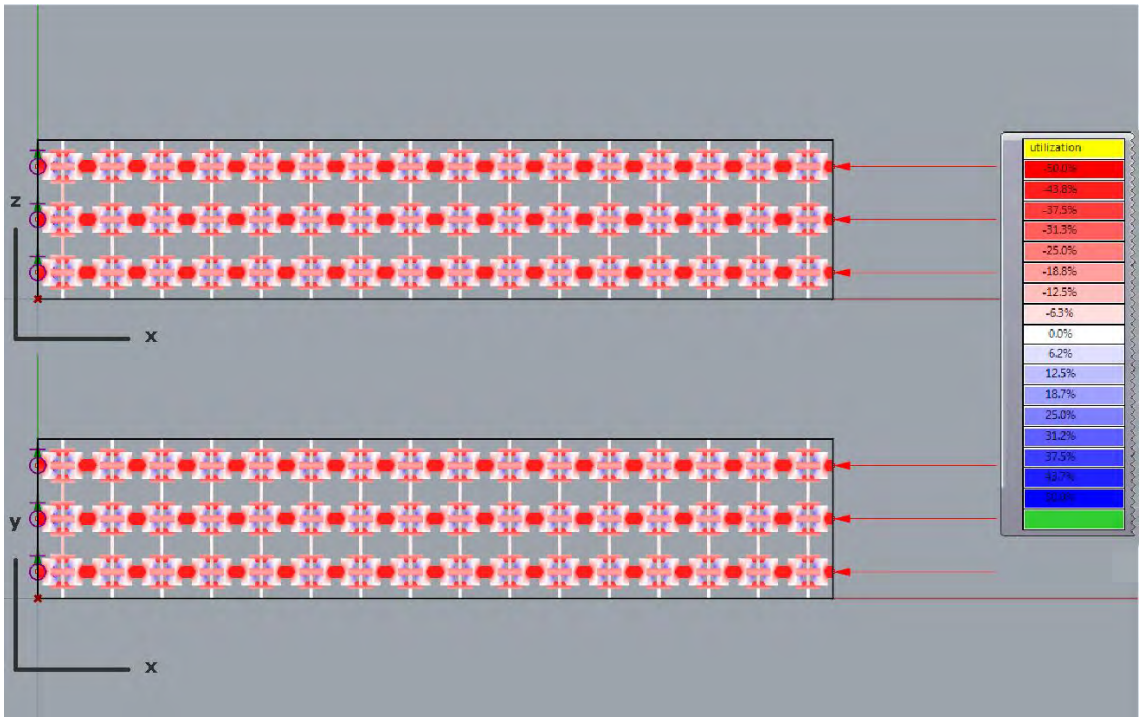

Figura 5.8: livelli di ottimizzazione degli elementi per la struttura definitiva.

Appare interessante notare come gli elementi trasversali della cella, orientati secondo l'asse y, lavorino comunque a flessione nonostante il carico sia puramente di compressione

5.1.4. Parallelepipedo 10x10x50 a cella BCC sottoposto a compressione.

Si consideri un caso simile a quello descritto al paragrafo precedente, con l'unica variante della cella BCC invece della GAM.

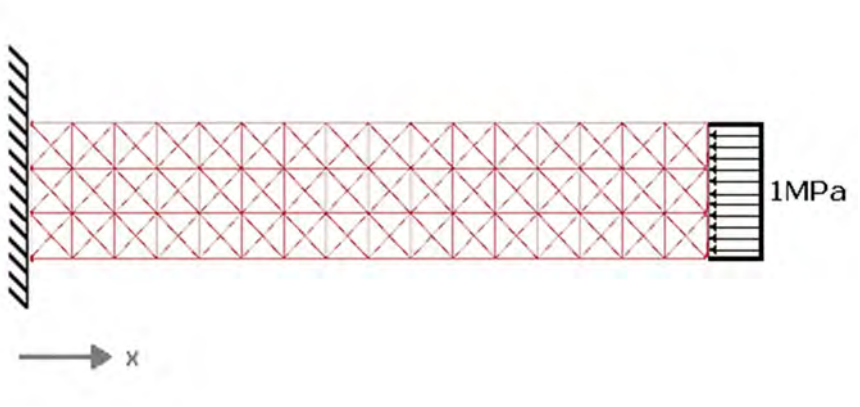

Figura 5.9: schema della struttura a cella BCC caricata e vincolata.

La struttura risultante ha le seguenti caratteristiche:

parallelepipedo 10x10x50					
cella	lato[mm]	N°celle	N° beam	N°nodi	R beam[mm]
BCC	3	144	1816	416	0.5

Tabella 5.11: caratteristiche della struttura schematizzata in figura 5.7.

Il materiale impiegato sia il poliammide PA2105 prodotto dalla EOS avente le caratteristiche meccaniche riportate in tabella:

PA 2105			
σsn[Mpa]	E[Gpa]	G[Gpa]	ρ[kg/m3]
54	1.85	0.748	950

Tabella 5.12: caratteristiche del materiale PA 2105.

Al modello si applichi l'algoritmo di ottimizzazione B, impostando R_{min} e R_{max} rispettivamente a 0,2 e 0,75mm, utilizzando come funzione di ottimizzazione la funzione radice quadrata e come funzione obbiettivo il massimo scostamento dal target UtiT, descritta al paragrafo 4.1.6.1.

iter.	NVG	Utimax	Utimin	Ins	Fobj%
0	0	19,8	0	0	100
1	0	27,5	0	0	100
2	0	34,3	4	0	92
3	0	38,8	4,5	0	91
4	0	41,6	7,3	0	85,6
5	0	43,3	8,1	0	83,8
10	0	46	27,2	0	45,6
20	0	48	42,4	0	15,1
29	0	48,7	45,3	0	9,42

Tabella 5.13: prospetto con le iterazioni effettuate dalla procedura B.

Dal prospetto si vede che è stata individuata la condizione di arresto alla ventinovesima iterazione, dove la funzione obbiettivo assume il valore di 9.42%

Si vede che la massima utilizzazione si stabilizza a valori molto prossimi a quella voluta, UtiT=50% già dalla ventesima iterazione. Anche l'utilizzazione minima ha un andamento crescente dalla decima iterazione in poi, fino a giungere a 45.3% all'iterazione 29.

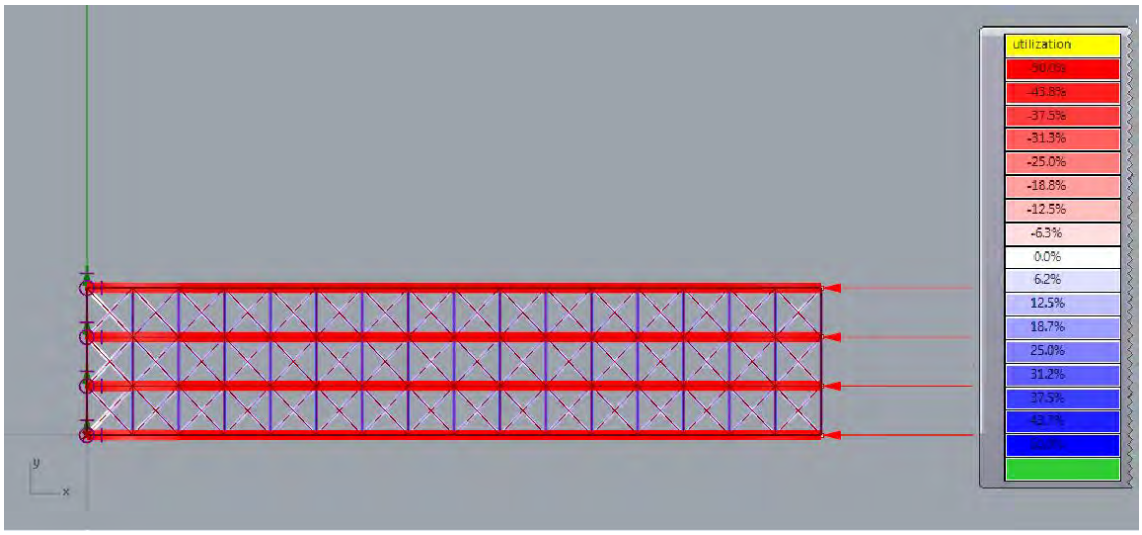

Figura 5.9: livelli di ottimizzazione degli elementi per la struttura definitiva.

Nella figura si vede la struttura fornita all'iterazione ventinove, si osserva la presenza di quattro colonne di elementi ingrossati che supportano il carico, anche nel piano x-z la situazione è identica.

5.2. Esempio di applicazione della procedura iterativa di ottimizzazione Beam e Shell: case study sfera.

Per testare le capacità della procedura iterativa di ottimizzazione Beam e Shell, si prenda in esame il caso di una sfera di raggio 50mm, vincolata sulla base e sottoposta ad un carico di compressione verticale di 750N[a]. La sfera, realizzata struttura cellulare con reticolo GAM di lato 10mm, ha un modello strutturale con le caratteristiche elencate nella tabella seguente:

Cella	N°celle	N°beam	N°shell	materiale
GAM	426	11502	7500	S235

Tabella 5.14: caratteristiche della struttura schematizzata in figura 5.10.

A tale modello strutturale sono applicati i vincoli, bloccando i nodi della *shell* appartenenti alla porzione inferiore della sfera e il carico, sui nodi della porzione superiore, come mostrato in figura.

[a]Al fine di semplificare la realizzazione della mesh su cui applicare gli elementi shell, la superficie sferica è stata approssimata con una simile ma costituita da elementi piani di forma quadrangolare.

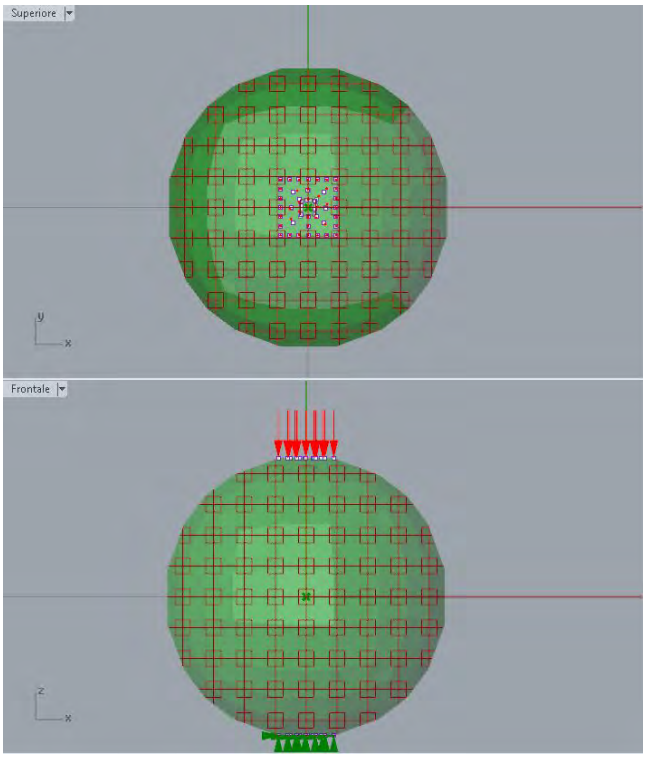

Figura 5.10: schema della struttura con cella GAM vincolata e caricata.

Il materiale della struttura è l'acciaio da costruzione S235JR avente le seguenti caratteristiche meccaniche:

S235JR			
σsn[Mpa]	E[Gpa]	G[Gpa]	ρ[kg/m3]
235	198	76	7800

Tabella 5.15: caratteristiche dell'acciaio S235JR.

La funzione di ottimizzazione delle *beam* per il calcolo dei nuovo raggio Rni è la radice quadrata del rapporto tra l'utilizzazione dell'elemento e l'utilizzazione target dell'equazione (4.16).

La funzione di ottimizzazione per le *shell* determina il nuovo spessore dell'elemento con una espressione simile alla (4.15), ma riferita agli spessori:

$$Sni = S * Ratio \quad (5.2)$$

Ratio è a sua volta una funzione composta dipendente dal rapporto tra l'utilizzazione dell'elemento e l'utilizzazione target uguale alla (5.1), riportata per comodità qui di seguito:

$$Ratio = 0.5 + 0.5\sqrt{1 - \left(\frac{Uti - UtiT}{UtiT}\right)^2}; \quad \frac{Uti}{UtiT} < 1$$

$$Ratio = 4\left(\frac{Uti}{UtiT}\right)^2 - 8\left(\frac{Uti}{UtiT}\right) + 5; \quad \frac{Uti}{UtiT} > 1$$

I valori entro cui possono variare i raggi delle *beam* sono compresi tra R_{max} e R_{min}, pari rispettivamente pari a 1,67 e 0,1mm. I valori entro cui possono variare gli spessori delle shell sono compresi tra S_{max}e S_{min}, pari rispettivamente a 2 e 0.2mm.

Con riferimento agli elementi aventi raggio superiore a R_{min}, la funzione obbiettivo nell'ottimizzazione della mesostruttura, è il massimo scostamento rispetto al target UtBT descritta al paragrafo 4.1.6.1.

Con riferimento allo schema di figura 4.12, la procedura incomincia generando una struttura con *beam* e *shell* a diametri e spessori uniformi pari a 3.3 e 0.2mm e opera in maniera iterativa modificando solo i diametri delle *beam* e mantenendo le *shell* allo spessore minimo fino che non sia stata raggiunta una condizione tale da soddisfare la condizione di terminazione, corrispondente a quanto riportato nell'intestazione di tabella 5.4.

beam					
iteraz.	el. NVG	el. ins	maxUti B	minUti B	fobj%
0	0	0	37,7	0	100
79	0	0	53,7	46,2	7,69

Tabella 5.13: prospetto con le iterazioni 0 e 79 effettuate per l'ottimizzazione della mesostruttura.

Come mostrato nella tabella 5.13, all'iterazione 79 si raggiunge la condizione di terminazione avendo soddisfatto la disequazione che concerne la Fobj e si considera perciò conclusa la fase di ottimizzazione della mesostruttura.
Passando all'ottimizzazione dell'esostruttura, sempre in riferimento allo schema di figura 4.12, la procedura assegna alle *shell* lo spessore massimo e incomincia la fase di ottimizzazione dell'esostruttura senza modificare la mesostruttura. La condizione di terminazione dell'ottimizzazione delle *shell* viene raggiunta quando la funzione FobjS, soddisfa la verifica FobjS<objS oppure all'insorgere di elementi *shell* insufficienti o *beam* NVG. Quest'ultima condizione deriva dal fatto che quando le *beam* non sono sovrautilizzate l'algoritmo non può intervenire modificando il loro diametro con lo scopo di abbassare il livello di utilizzazione.

shell							
iteraz.	el. NVG	maxUti b	minUti b	Sins	maxUti S	minUti S	FobjS%
80	0	18,9	0,1	0	15,6	0	100,00
81	0	27,9	0,1	0	19,5	0	100,00
82	0	36,6	1,1	0	22,9	0	100,00
83	0	44,1	2,7	0	26	0	100,00
84	0	50	0,3	0	28,6	2,3	95,40
85	0	53,6	2,1	0	30,8	2,2	95,60
86	0	54,9	6,2	0	32,9	5,5	89,00
87	0	54,1	8,8	0	34,8	8,5	83,00
88	0	52,5	12,5	0	37,6	10,5	79,00
89	0	51,5	16,3	0	40	12,5	75,00
90	0	50,5	15	0	42,9	13,8	72,40
91	0	52,2	11,9	0	45,5	17,2	65,60
92	1	55,3	12,3	0	45,6	16,1	67,80

Tabella 5.14: prospetto con le iterazioni da 80 e 92 effettuate per l'ottimizzazione dell'esostruttura.

La tabella CCCCC mostra quantitativamente i fenomeni descritti e si nota che all'iterazione 70 la struttura presenta i minimi valori di utilizzazione, dato che per tale configurazione le *shell* sono a spessore massimo. Col passare delle iterazioni, a causa dell'assottigliamento delle *shell* dovuto all'operazione di ottimizzazione, le utilizzazioni

della struttura tendono a crescere progressivamente. Le iterazioni proseguono fino all'82 dove c'è la comparsa di un elemento *beam* NVG.

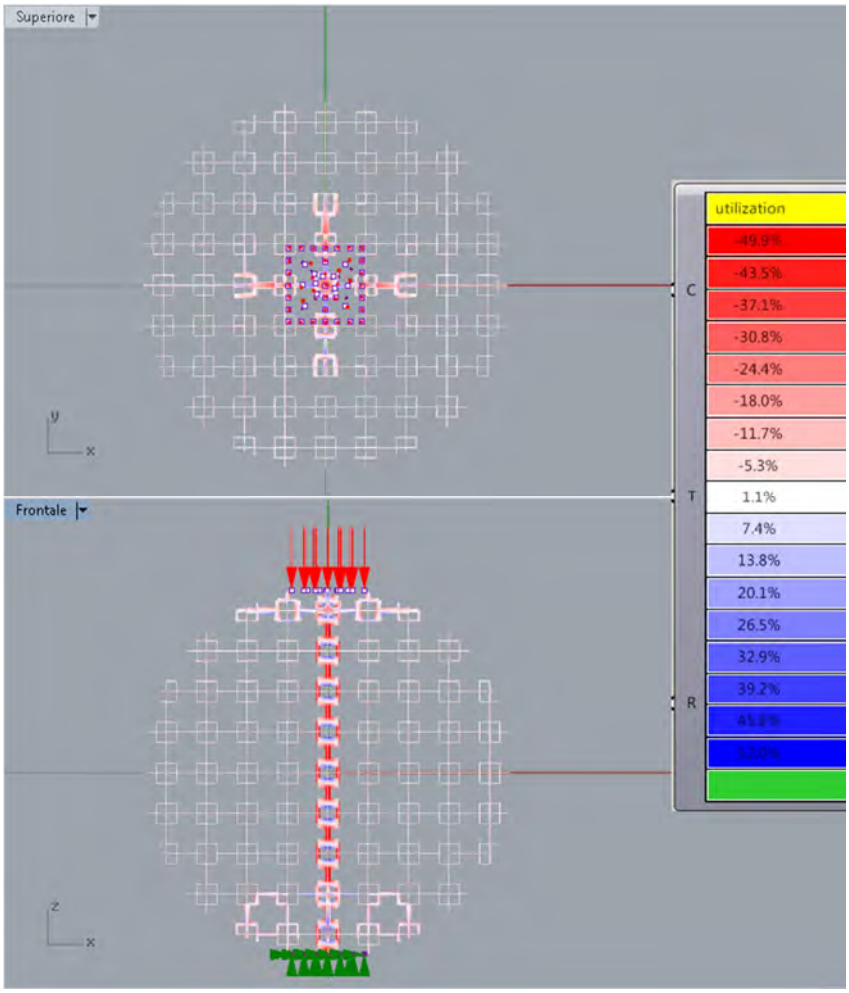

Figura 5.11: livelli di ottimizzazione degli elementi per la struttura definitiva.

In figura 5.11 si vede la distribuzione dei diametri degli elementi *beam* sulla struttura: è evidente l'azione di ottimizzazione svolta, data la comparsa di una "colonna" centrale di elementi ingrossati in corrispondenza di dove il carico va ad agire in maniera diretta e la riduzione al minimo degli elementi laterali poco caricati. Le aste ingrossate non sono limitate alla colonna di celle centrali, bensì si osservano anche in tutta la zona sollecitata attorno alle calotte. Questo comportamento indica l'interazione tra la gli elementi strutturali *beam* e *shell* della struttura.

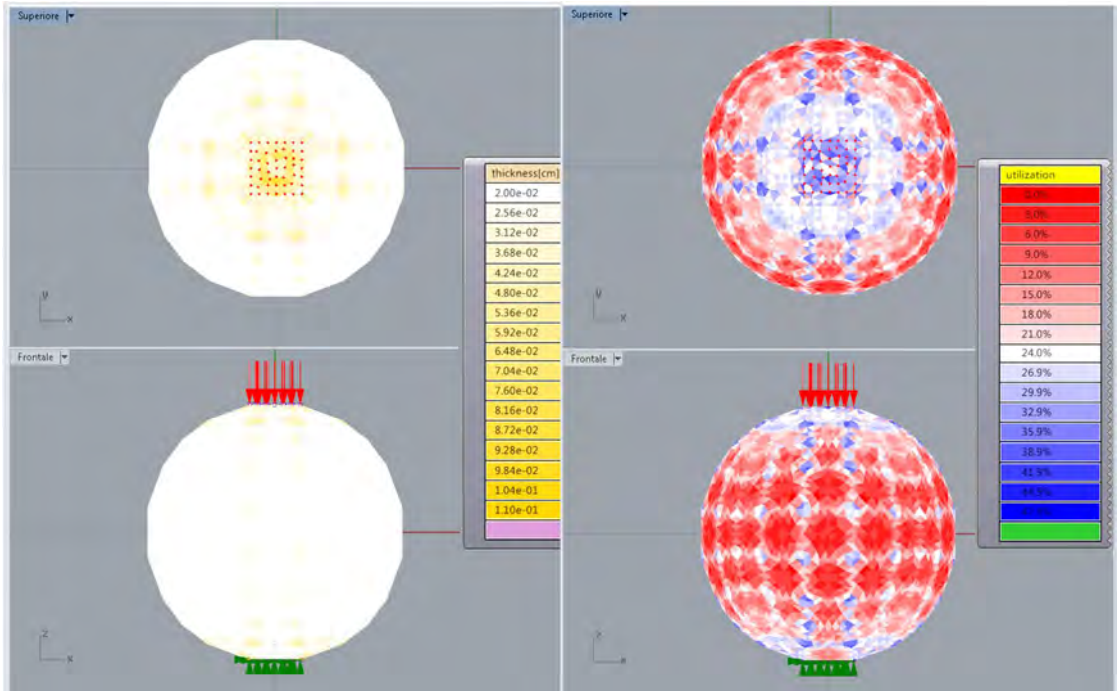

Figura 5.11: a sinistra la distribuzione degli spessori della shell, a destra i livelli di ottimizzazione degli elementi shell. L'immagine si riferisce alla struttura definitiva.

Per quanto riguarda degli elementi *shell*, si osserva che la procedura ha individuato nelle calotte la zona maggiormente sollecitata e qui ha mantenuto gli elementi a spessore maggiore. Si osserva inoltre la presenza di zone blu, che segnalano utilizzazione maggiore, in corrispondenza degli elementi posti in prossimità dei vertici delle giunture tra le superfici piane con cui si è approssimata la sfera. Probabilmente tale comportamento è da imputare al fatto che i bordi di giuntura agiscono da vincolo impedendo gli spostamenti reciproci delle superfici pane.

Per verificare se la presenza della mesostruttura sia utile ai fini di ottenete una struttura leggera ma il contempo rigida, si faccia un confronto, in termini di massimo spostamento registrato, tra la struttura ottenuta tramite la procedura B&S e una sfera cava di uguale massa, vincolata e caricata alla stessa maniera. Per determinare lo spessore S da assegnare alla sfera cava si consideri di determinare la superficie della sfera di diametro 100mm e di applicare uno spessore costante tale da generare un volume pari a quello della struttura ottimizzata.

Sfera cava					
m[kg]	ρ[kg/m3]	V mater[mm3]	R[mm]	Sup[mm2]	S[mm]
0,073	7800	9358,974	50	31415,927	0,30

Tabella 5.15: caratteristiche della struttura sfera cava.

Il massimo spostamento assoluto dei nodi si verifica per entrambe le tipologie di struttura in corrispondenza della zona di applicazione del carico. Nel caso di sfera cava: lo spostamento massimo è stimato in 6.94×10^{-2} cm, mentre nel caso della struttura cellulare è 5.27×10^{-3} cm. Il valore ottenuto nel caso di sfera cava non è realistico in quanto la struttura presenta valori di utilizzazione massima superiori al 300%. Ad ogni modo, il confronto tra una struttura che è in grado di sopportare il carico, come quella ottenuta dalla procedura B&S, contro una che non lo è, come quella cava, depone in favore della pima, dimostrando la sua efficacia.

6. Conclusioni e sviluppi futuri.

Per quanto concerne la procedura di iterativa di ottimizzazione beam, dai casi applicativi si osserva che l'algoritmo di ottimizzazione è in grado di convergere verso soluzioni migliori, che presentano livelli di utilizzazione degli elementi sempre più prossimi al target fissato UtiT, solamente fino alla comparsa degli elementi insufficienti[b]. Nel *case stu*dy 5.1.1, tabella 5.3, si dimostra che dopo la comparsa di elementi insufficienti le strutture generate presentano valori di utilizzazione massima degli elementi superiore al consentito e che al proseguire delle iterazioni il valore massimo aumenta costantemente. Qualora non si manifestasse l'insorgere di elementi insufficienti, la procedura riesce a migliorare sempre la struttura, raggiungendo una configurazione conforme agli obbiettivi prefissati, come si evince dai *case study* 5.1.2, 5.1.3 e 5.1.4. Confrontando il *case study* 5.1.1 con 5.1.2, che si riferiscono entrambi a strutture caricate a flessione dalla stessa forza verticale e vincolate in corrispondenza dei nodi della faccia opposta al carico, dove la prima impiega la cella elementare GAM e la seconda la cella elementare BCC, si vede come il fenomeno della comparsa di elementi insufficienti dipenda anche dal tipo di cella impiegata, dato che la struttura con cella BCC non manifesta il problema. Per quanto riguarda il carico a compressione, che a differenza di quello a flessione dato da una forza verticale, presenta un andamento dei parametri della sollecitazione costante lungo l'asse x, non si osserva la comparsa di elementi insufficienti in nessuno dei casi trattati.

La procedura iterativa di ottimizzazione beam e shell presenta le medesime problematiche esposte per la procedura per le beam e il problema degli elementi insufficienti affligge in modo particolare l'ottimizzazione dell'esostruttura. Considerando di applicare il carico direttamente ai nodo della *shell*, questo agisce direttamente su di essa, mentre sulla mesostruttura agisce indirettamente dato che coinvolge solamente i nodi comuni con l'esostruttura. Adottando una ottimizzazione simultanea, la generazione di nuove configurazioni strutturali potrebbe continuare fino alla comparsa di elementi insufficienti e per le ragioni esposte, saranno probabilmente

[b] Con elementi insufficienti si intendono quelli che presentano utilizzazione maggiore al 55% e che al contempo hanno raggio uguale al massimo consentito dall'algoritmo di ottimizzazione.

elementi *shell*. Quindi, essendo al durata delle due ottimizzazioni condizionata dalla parte di struttura che presenta per prima l'insorgere di elementi ins, l'ottimizzazione della mesostruttura verrebbe interrotta prematuramente.

Nonostante i problemi esposti, nel *case study* 5.2 è stata determinata una configurazione strutturali ottimizzata in grado di supportare i carichi assegnati.

La procedura basata su risolutore genetico, in linea di principio, potrebbe essere una via percorribile per la risoluzione dei problemi evidenziati nella procedura iterativa di ottimizzazione delle beam e nella procedura iterativa di ottimizzazione delle beam e Shell. Purtroppo sono stati riscontrate difficoltà nell'applicazione di tale procedura a casi simili a quelli presentati nel paragrafo 5.1. Innanzitutto non sono state trovate in letteratura indicazioni utili per un settaggio dei parametri del risolutore Galapagos, in particolare non si sono trovare correlazioni che esprimessero un legame tra il numero di variabili del problema, che indica la dimensione dello spazio di ricerca e la numerosità della popolazione. Infatti ci si aspetta che nel caso di problemi con molte variabili, come nell'applicazione a problemi come quelli illustrati nei *case study* del paragrafo 5.1, corrisponda una numerosità della popolazione adeguata.

La procedura basata su risolutore genetico è stata provata nel caso di ottimizzazione di strutture semplici, composte da circa una ventina di elementi e si è riscontrato come la procedura impiegasse moltissime generazioni, anche più di 400, far evolvere una popolazione di 50 individui verso una soluzione indicata come ottima. Inoltre, dato il carattere non deterministico di tale procedura, ripetendo lo svolgimento dell'ottimizzazione con le stesse condizioni di partenza si ottengono configurazioni finali differenti.

Il risolutore genetico offre la possibilità di impostare una delle soluzioni della generazione di partenza, gli altri individui che andranno a comporre la popolazione saranno invece ottenute in modo random. Di per se sarebbe un modo per inserire un individuo dominante i cui geni condizioneranno sicuramente le generazioni successive velocizzando la convergenza. Purtroppo l'impostazione di tale soluzione è manuale ed implica il posizionare un a lunga serie di cursori su valori numerici precisi, che si traduce in un lavoro impossibile nel caso in cui si operi con più di una cinquantina di variabili.

Individuare una maniera per impostare in modo automatico i cursori del “Gene Pool” sui valori che definiscono la soluzione voluta permetterebbe di integrare la procedura B e la procedura basata su risolutore genetico. Tale integrazione potrebbe permettere di risolvere i problemi evidenziati nella procedura iterativa di ottimizzazione delle beam, infatti dopo aver determinato la configurazione che presenta la comparsa di elementi insufficienti, si potrebbe pensare di inserirla come soluzione iniziale nella “Gene Pool”, facendo agire a questo punto la procedura basata su risolutore genetico.

Un altra soluzione al problema della comparsa di elementi insufficienti sarebbe proporre un algoritmo che, alla comparsa di elementi insufficienti, cercasse di ridurre il carico agente dell’elemento insufficiente redistribuendolo agli elementi vicini. Per fare ciò dovrebbe svincolare questi elementi vicini, identificati come quelli che condividono un nodo con l’elemento insufficiente, dalla logica ottimizzatoria di raggiungere l’utilizzazione target.

Dato che il modello virtuale è composto da molti cilindri compenetrati e non da un unico solido corrispondente alla loro unione booleana, la stima del volume della struttura da parte di Karamba è per eccesso, con precisione decrescente al crescere della dimensione degli oggetti che si compenetrano. Per tale ragione nei *case study* si è preferito non fornire il dato relativo al volume della struttura ottenuta. Anche prevedendo di eseguire l’unione booleana dei solidi a procedura conclusa per mezzo del software Rhinoceros5, l’operazione non va a buon fine a causa del gran numero di solidi coinvolti nell’operazione. Di conseguenza ulteriori sviluppi devono riguardare anche la definizione di una metodologia che permetta di ottenere il solido risultante dall’unione booleana dei cilindri che compongono il modello virtuale. Una soluzione potrebbe essere quella di operare per step intermedi costituendo prima dei solidi formati da elementi che convergono in un nodo e poi proseguire unendo uno per volta i solidi ottenuti fino a che si sia costituita l’unione booleana voluta.

Appendice

Create Model Script

```
import rhinoscriptsyntax as rs
from Rhino.Geometry import Point3d
import clr
clr.AddReferenceToFileAndPath(r'H:\LibSC\ipython\alglibnet2.dll')
import xalglib as alg
import System.Drawing.Color as Color
import math

def CubicCell(base1, base2, altezza, StartPt = None):
    """
    Crea gli spigoli di un parallelepipedo di dimensioni
basel,base2,altezza
    """

    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError("Start Pt is not valid")

    X = pt.X
    Y = pt.Y
    Z = pt.Z

    vertex = alg.create_real_vector(8)
    vertex[0] = [X, Y, Z]
    vertex[1] = [X, Y + base2, Z]
    vertex[2] = [X + base1, Y + base2, Z]
    vertex[3] = [X + base1, Y, Z]
    vertex[4] = [X, Y, Z + altezza]
    vertex[5] = [X, Y + base2, Z + altezza]
    vertex[6] = [X + base1, Y + base2, Z + altezza]
    vertex[7] = [X + base1, Y , Z + altezza]

    rs.EnableRedraw(False)
    i = 0
    j = 0
    UnionVertex = []
    for v in vertex:
        i = i + 1
        a = rs.coerce3dpoint(v)
        for p in vertex:
            j = j + 1
            if j > i:
                b = rs.coerce3dpoint(p)
                if a.X == b.X :
                    if a.Y == b.Y:
                        uv = [a,b]
                        UnionVertex.append(uv)
                    else :
                        if a.Z == b.Z:
```

```
                            uv = [a,b]
                            UnionVertex.append(uv)
                    if (a.Y == b.Y) and (a.Z == b.Z):
                        uv = [a,b]
                        UnionVertex.append(uv)
            j = 0
        rs.EnableRedraw(True)
        return UnionVertex, vertex

def BodyCenterCubic(basex, basey, basez, StartPt = None):

    """
    Crea la cella elementare Cubica a corpo centrato
    """
    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError ("Start Pt is not valid")
    X = pt.X
    Y = pt.Y
    Z = pt.Z

    vertex = alg.create_real_vector(9)
    vertex[0] = Point3d(X, Y, Z)
    vertex[1] = Point3d(X+basex, Y, Z)
    vertex[2] = Point3d(X, Y+basey, Z)
    vertex[3] = Point3d(X+basex, Y+basey, Z)
    vertex[4] = Point3d(X+(0.5*basex), Y+(0.5*basey), Z+(0.5*basez))
    vertex[5] = Point3d(X, Y, Z+basez)
    vertex[6] = Point3d(X+basex, Y, Z+basez)
    vertex[7] = Point3d(X, Y+basey, Z+basez)
    vertex[8] = Point3d(X+basex, Y+basey, Z+basez)

    rs.EnableRedraw(False)

    UnionVertex = []
    UnionVertex.append ([vertex[0],vertex[1]])
    UnionVertex.append ([vertex[0],vertex[2]])
    UnionVertex.append ([vertex[1],vertex[3]])
    UnionVertex.append ([vertex[2],vertex[3]])
    UnionVertex.append ([vertex[0],vertex[5]])
    UnionVertex.append ([vertex[1],vertex[6]])
    UnionVertex.append ([vertex[2],vertex[7]])
    UnionVertex.append ([vertex[3],vertex[8]])
    UnionVertex.append ([vertex[5],vertex[6]])
    UnionVertex.append ([vertex[5],vertex[7]])
    UnionVertex.append ([vertex[6],vertex[8]])
    UnionVertex.append ([vertex[7],vertex[8]])
    UnionVertex.append ([vertex[0],vertex[4]])
    UnionVertex.append ([vertex[1],vertex[4]])
    UnionVertex.append ([vertex[2],vertex[4]])
    UnionVertex.append ([vertex[3],vertex[4]])
    UnionVertex.append ([vertex[4],vertex[5]])
    UnionVertex.append ([vertex[4],vertex[6]])
    UnionVertex.append ([vertex[4],vertex[7]])
```

```
    UnionVertex.append ([vertex[4],vertex[8]])
    rs.EnableRedraw(True)
    print vertex[4]
    return UnionVertex, vertex

def ReinforcedBodyCenterCubic(basex, basey, basez, StartPt = None):
    """
    Crea la cella elementare Cubica a corpo centrato
    """

    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError("Start Pt is not valid")

    X = pt.X
    Y = pt.Y
    Z = pt.Z

    #body center cubic
    vertex = alg.create_real_vector(15)
    vertex[0] = Point3d(X, Y, Z)
    vertex[1] = Point3d(X+basex, Y , Z)
    vertex[2] = Point3d(X, Y+basey, Z)
    vertex[3] = Point3d(X+basex, Y+basey, Z)
    vertex[4] = Point3d(X+(0.5*basex), Y+(0.5*basey), Z+(0.5*basez))
    vertex[5] = Point3d(X, Y, Z+basez)
    vertex[6] = Point3d(X+basex, Y , Z+basez)
    vertex[7] = Point3d(X, Y+basey, Z+basez)
    vertex[8] = Point3d(X+basex, Y+basey, Z+basez)
    #reinforced
    vertex[9] = Point3d(X+(0.5*basex), Y+(0.5*basey), Z-(0.5*basez))
    vertex[10] = Point3d(X+(0.5*basex), Y-(0.5*basey), Z+(0.5*basez))
    vertex[11] = Point3d(X-(0.5*basex), Y+(0.5*basey), Z+(0.5*basez))
    vertex[12] = Point3d(X+(1.5*basex), Y+(0.5*basey), Z+(0.5*basez))
    vertex[13] = Point3d(X+(0.5*basex), Y+(1.5*basey), Z+(0.5*basez))
    vertex[14] = Point3d(X+(0.5*basex), Y+(0.5*basey), Z+(1.5*basez))

    #body center cubic
    UnionVertex = []
    UnionVertex.append([vertex[0],vertex[1]])
    UnionVertex.append([vertex[0],vertex[2]])
    UnionVertex.append([vertex[1],vertex[3]])
    UnionVertex.append([vertex[2],vertex[3]])
    UnionVertex.append([vertex[0],vertex[5]])
    UnionVertex.append([vertex[1],vertex[6]])
    UnionVertex.append([vertex[2],vertex[7]])
    UnionVertex.append([vertex[3],vertex[8]])
    UnionVertex.append([vertex[5],vertex[6]])
    UnionVertex.append([vertex[5],vertex[7]])
    UnionVertex.append([vertex[6],vertex[8]])
    UnionVertex.append([vertex[7],vertex[8]])
    UnionVertex.append([vertex[0],vertex[4]])
    UnionVertex.append([vertex[1],vertex[4]])
    UnionVertex.append([vertex[2],vertex[4]])
```

```
    UnionVertex.append([vertex[3],vertex[4]])
    UnionVertex.append([vertex[4],vertex[5]])
    UnionVertex.append([vertex[4],vertex[6]])
    UnionVertex.append([vertex[4],vertex[7]])
    UnionVertex.append([vertex[4],vertex[8]])
    #reinfoced
    UnionVertex.append([vertex[9],vertex[4]])
    UnionVertex.append([vertex[10],vertex[4]])
    UnionVertex.append([vertex[11],vertex[4]])
    UnionVertex.append([vertex[4],vertex[12]])
    UnionVertex.append([vertex[4],vertex[13]])
    UnionVertex.append([vertex[4],vertex[14]])

    return UnionVertex, vertex

def WallachGibson2001(bx, bt, h, StartPt = None):
    """
    Crea la cella elementare proposta da Wallach, Gibson
    "Mechanical behaviour of three-dimensional truss material" 2001
    """

    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError("Start Pt is not valid")

    X = pt.X
    Y = pt.Y
    Z = pt.Z

    vertex = alg.create_real_vector(15)
    vertex[0] = Point3d(X, Y, Z)
    vertex[1] =  Point3d(X + bx/2, Y    , Z)
    vertex[2] = Point3d(X +  bx, Y , Z)
    vertex[3] = Point3d(X, Y + bt/2, Z)
    vertex[4] = Point3d(X + bx, Y + bt/2, Z)
    vertex[5] = Point3d(X, Y + bt, Z)
    vertex[6] =  Point3d(X + bx/2, Y    + bt, Z)
    vertex[7] =  Point3d(X + bx, Y +    bt, Z)
    vertex[8] = Point3d(X, Y, Z + h)
    vertex[9] = Point3d(X + bx/2, Y , Z  + h)
    vertex[10] = Point3d(X + bx, Y , Z + h)
    vertex[11] = Point3d(X + bx/2 , Y +  bt/2, Z + h)
    vertex[12] = Point3d(X, Y + bt, Z +  h)
    vertex[13] = Point3d(X + bx/2, Y + bt, Z + h)
    vertex[14] = Point3d(X + bx, Y + bt, Z + h)

    rs.EnableRedraw(False)
    UnionVertex = []
    UnionVertex.append([vertex[0],vertex[3]])
    UnionVertex.append([vertex[1],vertex[3]])
    UnionVertex.append([vertex[1],vertex[6]])
    UnionVertex.append([vertex[1],vertex[4]])
    UnionVertex.append([vertex[2],vertex[4]])
    UnionVertex.append([vertex[3],vertex[5]])
```

```
    UnionVertex.append([vertex[3],vertex[6]])
    UnionVertex.append([vertex[4],vertex[6]])
    UnionVertex.append([vertex[4],vertex[7]])
    UnionVertex.append([vertex[1],vertex[8]])
    UnionVertex.append([vertex[1],vertex[10]])
    UnionVertex.append([vertex[3],vertex[8]])
    UnionVertex.append([vertex[1],vertex[11]])
    UnionVertex.append([vertex[4],vertex[10]])
    UnionVertex.append([vertex[3],vertex[11]])
    UnionVertex.append([vertex[4],vertex[11]])
    UnionVertex.append([vertex[3],vertex[12]])
    UnionVertex.append([vertex[6],vertex[11]])
    UnionVertex.append([vertex[4],vertex[14]])
    UnionVertex.append([vertex[6],vertex[12]])
    UnionVertex.append([vertex[6],vertex[14]])
    UnionVertex.append([vertex[8],vertex[12]])
    UnionVertex.append([vertex[8],vertex[11]])
    UnionVertex.append([vertex[9],vertex[11]])
    UnionVertex.append([vertex[10],vertex[11]])
    UnionVertex.append([vertex[10],vertex[14]])
    UnionVertex.append([vertex[11],vertex[12]])
    UnionVertex.append([vertex[11],vertex[13]])
    UnionVertex.append([vertex[11],vertex[14]])
    rs.EnableRedraw(True)
    return UnionVertex, vertex

def OctectTruss(bx, by, h, StartPt = None):
    """
    Crea la cel]a elementare OctetTruss
    "Mechanical behaviour of three-dimensional truss material" 2001
    """

    if StartPt == None :
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None :
        raise NameError("Start Pt is not valid")

    X = pt.X
    Y = pt.Y
    Z = pt.Z

    vertex = alg.create_real_vector(14)
    vertex[0] = Point3d(X, Y, Z)
    vertex[1] = Point3d(X+bx, Y , Z)
    vertex[2] = Point3d(X+bx/2, Y+by/2 , Z)
    vertex[3] = Point3d(X, Y+by, Z)
    vertex[4] = Point3d(X+bx, Y+by, Z)
    vertex[5] = Point3d(X+bx/2, Y, Z+h/2)
    vertex[6] = Point3d(X, Y+by/2, Z+h/2)
    vertex[7] = Point3d(X+bx, Y+by/2, Z+h/2)
    vertex[8] = Point3d(X+bx/2, Y+by, Z+h/2)
    vertex[9] = Point3d(X, Y, Z+h)
    vertex[10] = Point3d(X+bx, Y , Z+h)
    vertex[11] = Point3d(X+bx/2, Y+by/2 , Z+h)
    vertex[12] = Point3d(X, Y+by, Z+h)
```

```
    vertex[13] = Point3d(X+bx, Y+by, Z+h)

    rs.EnableRedraw(False)
    UnionVertex = []
    UnionVertex.append([vertex[0],vertex[2]])
    UnionVertex.append([vertex[1],vertex[2]])
    UnionVertex.append([vertex[2],vertex[3]])
    UnionVertex.append([vertex[2],vertex[4]])
    UnionVertex.append([vertex[0],vertex[5]])
    UnionVertex.append([vertex[1],vertex[5]])
    UnionVertex.append([vertex[0],vertex[6]])
    UnionVertex.append([vertex[1],vertex[7]])
    UnionVertex.append([vertex[3],vertex[6]])
    UnionVertex.append([vertex[4],vertex[7]])
    UnionVertex.append([vertex[3],vertex[8]])
    UnionVertex.append([vertex[4],vertex[8]])
    UnionVertex.append([vertex[5],vertex[9]])
    UnionVertex.append([vertex[5],vertex[10]])
    UnionVertex.append([vertex[6],vertex[9]])
    UnionVertex.append([vertex[7],vertex[10]])
    UnionVertex.append([vertex[6],vertex[12]])
    UnionVertex.append([vertex[7],vertex[13]])
    UnionVertex.append([vertex[8],vertex[12]])
    UnionVertex.append([vertex[8],vertex[13]])
    UnionVertex.append([vertex[9],vertex[11]])
    UnionVertex.append([vertex[10],vertex[11]])
    UnionVertex.append([vertex[11],vertex[12]])
    UnionVertex.append([vertex[11],vertex[13]])
    UnionVertex.append([vertex[2],vertex[5]])
    UnionVertex.append([vertex[2],vertex[6]])
    UnionVertex.append([vertex[2],vertex[7]])
    UnionVertex.append([vertex[2],vertex[8]])
    UnionVertex.append([vertex[5],vertex[6]])
    UnionVertex.append([vertex[5],vertex[7]])
    UnionVertex.append([vertex[6],vertex[8]])
    UnionVertex.append([vertex[7],vertex[8]])
    UnionVertex.append([vertex[5],vertex[11]])
    UnionVertex.append([vertex[6],vertex[11]])
    UnionVertex.append([vertex[7],vertex[11]])
    UnionVertex.append([vertex[8],vertex[11]])
    rs.EnableRedraw(True)
    return UnionVertex, vertex

def TSC(bx, by, h, StartPt = None):
    """
    Crea la cella elementare Translated Simple Cubic
    "Regular, low density cellular structures - rapid prototyping,
    numerical simulation, mechanical testing" 2004
    """

    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError("Start Pt is not valid")
```

```
X = pt.X
Y = pt.Y
Z = pt.Z

vertex = alg.create_real_vector(26)
vertex[0] = Point3d(X, Y, Z)
vertex[1] = Point3d(X, Y+by/2, Z)
vertex[2] = Point3d(X, Y+by/2, Z+h/4)
vertex[3] = Point3d(X, Y+by, Z+h/4)
vertex[4] = Point3d(X, Y, Z+h/2)
vertex[5] = Point3d(X, Y+by/2, Z+h/2)
vertex[6] = Point3d(X, Y+by/2, Z+h*3/4)
vertex[7] = Point3d(X, Y+by, Z+h*3/4)
vertex[8] = Point3d(X, Y, Z+h)
vertex[9] = Point3d(X, Y+by/2, Z+h)
#FINE PRIMA FACCIA
vertex[10] = Point3d(X+bx/2, Y, Z)
vertex[11] = Point3d(X+bx/2, Y+by/2, Z)
vertex[12] = Point3d(X+bx/2, Y, Z+h/4)
vertex[13] = Point3d(X+bx/2, Y+by/2, Z+h/4)
vertex[14] = Point3d(X+bx/2, Y+by, Z+h/4)
vertex[15] = Point3d(X+bx/2, Y, Z+h/2)
vertex[16] = Point3d(X+bx/2, Y+by/2, Z+h/2)
vertex[17] = Point3d(X+bx/2, Y, Z+h*3/4)
vertex[18] = Point3d(X+bx/2, Y+by/2, Z+h*3/4)
vertex[19] = Point3d(X+bx/2, Y+by, Z+h*3/4)
vertex[20] = Point3d(X+bx/2, Y, Z+h)
vertex[21] = Point3d(X+bx/2, Y+by/2, Z+h)
#Fine seconda faccia
vertex[22] = Point3d(X+bx, Y, Z+h/4)
vertex[23] = Point3d(X+bx, Y+by/2, Z+h/4)
vertex[24] = Point3d(X+bx, Y, Z+h*3/4)
vertex[25] = Point3d(X+bx, Y+by/2, Z+h*3/4)
#Fine terza faccia


UnionVertex = []
UnionVertex.append([vertex[0],vertex[1]])
UnionVertex.append([vertex[1],vertex[2]])
UnionVertex.append([vertex[2],vertex[5]])
UnionVertex.append([vertex[0],vertex[4]])
UnionVertex.append([vertex[4],vertex[5]])
UnionVertex.append([vertex[4],vertex[8]])
UnionVertex.append([vertex[5],vertex[9]])
UnionVertex.append([vertex[5],vertex[6]])
UnionVertex.append([vertex[6],vertex[9]])
UnionVertex.append([vertex[2],vertex[3]])
UnionVertex.append([vertex[3],vertex[7]])
UnionVertex.append([vertex[6],vertex[7]])
#fine prima faccia
UnionVertex.append([vertex[0],vertex[10]])
UnionVertex.append([vertex[1],vertex[11]])
UnionVertex.append([vertex[2],vertex[13]])
UnionVertex.append([vertex[3],vertex[14]])
UnionVertex.append([vertex[4],vertex[15]])
UnionVertex.append([vertex[5],vertex[16]])
```

```
    UnionVertex.append([vertex[6],vertex[18]])
    UnionVertex.append([vertex[7],vertex[19]])
    UnionVertex.append([vertex[8],vertex[20]])
    UnionVertex.append([vertex[9],vertex[21]])
    #fine connessioni prima e seconda faccia
    UnionVertex.append([vertex[10],vertex[11]])
    UnionVertex.append([vertex[10],vertex[12]])
    UnionVertex.append([vertex[12],vertex[13]])
    UnionVertex.append([vertex[11],vertex[13]])
    UnionVertex.append([vertex[12],vertex[15]])
    UnionVertex.append([vertex[13],vertex[16]])
    UnionVertex.append([vertex[15],vertex[16]])
    UnionVertex.append([vertex[15],vertex[17]])
    UnionVertex.append([vertex[16],vertex[18]])
    UnionVertex.append([vertex[17],vertex[18]])
    UnionVertex.append([vertex[17],vertex[20]])
    UnionVertex.append([vertex[18],vertex[21]])
    UnionVertex.append([vertex[20],vertex[21]])
    UnionVertex.append([vertex[13],vertex[14]])
    UnionVertex.append([vertex[18],vertex[19]])
    UnionVertex.append([vertex[14],vertex[19]])
    #fine seconda faccia
    UnionVertex.append([vertex[12],vertex[22]])
    UnionVertex.append([vertex[13],vertex[23]])
    UnionVertex.append([vertex[17],vertex[24]])
    UnionVertex.append([vertex[18],vertex[25]])
    #fine connessioni seconda e terza faccia
    UnionVertex.append([vertex[22],vertex[23]])
    UnionVertex.append([vertex[22],vertex[24]])
    UnionVertex.append([vertex[23],vertex[25]])
    UnionVertex.append([vertex[24],vertex[25]])
    #fine terza faccia
    return UnionVertex, vertex

def GAMseparata(bx, by, h, StartPt = None):
    """
    Crea la cella elementare Gibson-Ashby Modificata
    "Elastic properties of model random three-dimensional open-cell
    solids" 2002
    """

    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError("Start Pt is not valid")

    X = pt.X
    Y = pt.Y
    Z = pt.Z

    vertex = alg.create_real_vector(24)
    vertex[0] = Point3d(X, Y+by/2, Z+h/2)
    #fine prima faccia
    vertex[1] = Point3d(X+bx/4, Y+by/2, Z+h/4)
    vertex[2] = Point3d(X+bx/4, Y+by/4, Z+h/2)
```

```
vertex[3] = Point3d(X+bx/4, Y+by/2, Z+h/2)
vertex[4] = Point3d(X+bx/4, Y+by*3/4, Z+h/2)
vertex[5] = Point3d(X+bx/4, Y+by/2, Z+h*3/4)
#fine seconda faccia
vertex[6] = Point3d(X+bx/2, Y+by/2, Z)
vertex[7] = Point3d(X+bx/2, Y+by/4, Z+h/4)
vertex[8] = Point3d(X+bx/2, Y+by/2, Z+h/4)
vertex[9] = Point3d(X+bx/2,     Y+by*3/4, Z+h/4)
vertex[10] = Point3d(X+bx/2, Y, Z+h/2)
vertex[11] = Point3d(X+bx/2, Y+by/4, Z+h/2)
vertex[12] = Point3d(X+bx/2, Y+by*3/4, Z+h/2)
vertex[13] = Point3d(X+bx/2, Y+by, Z+h/2)
vertex[14] = Point3d(X+bx/2, Y+by/4, Z+h*3/4)
vertex[15] = Point3d(X+bx/2, Y+by/2, Z+h*3/4)
vertex[16] = Point3d(X+bx/2, Y+by*3/4, Z+h*3/4)
vertex[17] = Point3d(X+bx/2, Y+by/2, Z+h)
#Fine terza faccia
vertex[18] = Point3d(X+bx*3/4, Y+by/2, Z+h/4)
vertex[19] = Point3d(X+bx* 3/4, Y+by/4, Z+h/2)
vertex[20] = Point3d(X+bx*3/4, Y+by/2 , Z+h/2)
vertex[21] = Point3d(X+bx*3/4, Y+by*3/4 , Z+h/2)
vertex[22] = Point3d(X+bx*3/4, Y+by/2, Z+h*3/4)
#fine quarta faccia
vertex[23] = Point3d(X+bx, Y+by/2 , Z+h/2)

UnionVertex = []
UnionVertex.append([vertex[0],vertex[3]])
#fine connessione prima e seconda faccia
UnionVertex.append([vertex[1],vertex[3]])
UnionVertex.append([vertex[2],vertex[3]])
UnionVertex.append([vertex[3],vertex[4]])
UnionVertex.append([vertex[3],vertex[5]])
#fine faccia 2
UnionVertex.append([vertex[1],vertex[8]])
UnionVertex.append([vertex[2],vertex[11]])
UnionVertex.append([vertex[4],vertex[12]])
UnionVertex.append([vertex[5],vertex[15]])
#fine connessione seconda e terza faccia
UnionVertex.append([vertex[6],vertex[8]])
UnionVertex.append([vertex[7],vertex[8]])
UnionVertex.append([vertex[8],vertex[9]])
UnionVertex.append([vertex[10],vertex[11]])
UnionVertex.append([vertex[7],vertex[11]])
UnionVertex.append([vertex[9],vertex[12]])
UnionVertex.append([vertex[12],vertex[13]])
UnionVertex.append([vertex[11],vertex[14]])
UnionVertex.append([vertex[14],vertex[15]])
UnionVertex.append([vertex[15],vertex[16]])
UnionVertex.append([vertex[12],vertex[16]])
UnionVertex.append([vertex[15],vertex[17]])
#fine terza faccia
UnionVertex.append([vertex[8],vertex[18]])
UnionVertex.append([vertex[11],vertex[19]])
UnionVertex.append([vertex[12],vertex[21]])
UnionVertex.append([vertex[15],vertex[22]])
#fine connessioni terza e quarta faccia
```

```
    UnionVertex.append([vertex[18],vertex[20]])
    UnionVertex.append([vertex[19],vertex[20]])
    UnionVertex.append([vertex[20],vertex[21]])
    UnionVertex.append([vertex[20],vertex[22]])
    #fine quarta faccia
    UnionVertex.append([vertex[20],vertex[23]])
    return UnionVertex, vertex

def GAM(bx, by, h, StartPt = None):
    """
    Crea la cella elementare Gibson-Ashby Modificata
    "Elastic properties of model random three-dimensional open-cell
    solids" 2002
    """

    if StartPt == None:
        StartPt = [0.,0.,0.]
    pt = rs.coerce3dpoint(StartPt)
    if pt == None:
        raise NameError("Start Pt is not valid")

    X = pt.X
    Y = pt.Y
    Z = pt.Z

    vertex = alg.create_real_vector(24)
    vertex[0] = Point3d(X-bx/4, Y+by/2, Z+h/2)
    #fine prima faccia
    vertex[1] = Point3d(X+bx/4, Y+by/2, Z+h/4)
    vertex[2] = Point3d(X+bx/4, Y+by/4, Z+h/2)
    vertex[3] = Point3d(X+bx/4, Y+by/2, Z+h/2)
    vertex[4] = Point3d(X+bx/4, Y+by*3/4, Z+h/2)
    vertex[5] = Point3d(X+bx/4, Y+by/2, Z+h*3/4)
    #fine seconda faccia
    vertex[6] = Point3d(X+bx/2, Y+by/2, Z-h/4)
    vertex[7] = Point3d(X+bx/2, Y+by/4, Z+h/4)
    vertex[8] = Point3d(X+bx/2, Y+by/2, Z+h/4)
    vertex[9] = Point3d(X+bx/2,     Y+by*3/4, Z+h/4)
    vertex[10] = Point3d(X+bx/2, Y-by/4, Z+h/2)
    vertex[11] = Point3d(X+bx/2, Y+by/4, Z+h/2)
    vertex[12] = Point3d(X+bx/2, Y+by*3/4, Z+h/2)
    vertex[13] = Point3d(X+bx/2, Y+by+by/4, Z+h/2)
    vertex[14] = Point3d(X+bx/2, Y+by/4, Z+h*3/4)
    vertex[15] = Point3d(X+bx/2, Y+by/2, Z+h*3/4)
    vertex[16] = Point3d(X+bx/2, Y+by*3/4, Z+h*3/4)
    vertex[17] = Point3d(X+bx/2, Y+by/2, Z+h+h/4)
    #Fine terza faccia
    vertex[18] = Point3d(X+bx*3/4, Y+by/2, Z+h/4)
    vertex[19] = Point3d(X+bx* 3/4, Y+by/4, Z+h/2)
    vertex[20] = Point3d(X+bx*3/4, Y+by/2 , Z+h/2)
    vertex[21] = Point3d(X+bx*3/4, Y+by*3/4 , Z+h/2)
    vertex[22] = Point3d(X+bx*3/4, Y+by/2, Z+h*3/4)
    #fine quarta faccia
    vertex[23] = Point3d(X+bx+bx/4, Y+by/2 , Z+h/2)

    UnionVertex = []
```

```
    UnionVertex.append([vertex[0],vertex[3]])
    #fine connessione prima e seconda faccia
    UnionVertex.append([vertex[1],vertex[3]])
    UnionVertex.append([vertex[2],vertex[3]])
    UnionVertex.append([vertex[3],vertex[4]])
    UnionVertex.append([vertex[3],vertex[5]])
    #fine faccia 2
    UnionVertex.append([vertex[1],vertex[8]])
    UnionVertex.append([vertex[2],vertex[11]])
    UnionVertex.append([vertex[4],vertex[12]])
    UnionVertex.append([vertex[5],vertex[15]])
    #fine connessione seconda e terza faccia
    UnionVertex.append([vertex[6],vertex[8]])
    UnionVertex.append([vertex[7],vertex[8]])
    UnionVertex.append([vertex[8],vertex[9]])
    UnionVertex.append([vertex[10],vertex[11]])
    UnionVertex.append([vertex[7],vertex[11]])
    UnionVertex.append([vertex[9],vertex[12]])
    UnionVertex.append([vertex[12],vertex[13]])
    UnionVertex.append([vertex[11],vertex[14]])
    UnionVertex.append([vertex[14],vertex[15]])
    UnionVertex.append([vertex[15],vertex[16]])
    UnionVertex.append([vertex[12],vertex[16]])
    UnionVertex.append([vertex[15],vertex[17]])
    #fine terza faccia
    UnionVertex.append([vertex[8],vertex[18]])
    UnionVertex.append([vertex[11],vertex[19]])
    UnionVertex.append([vertex[12],vertex[21]])
    UnionVertex.append([vertex[15],vertex[22]])
    #fine connessioni terza e quarta faccia
    UnionVertex.append([vertex[18],vertex[20]])
    UnionVertex.append([vertex[19],vertex[20]])
    UnionVertex.append([vertex[20],vertex[21]])
    UnionVertex.append([vertex[20],vertex[22]])
    #fine quarta faccia
    UnionVertex.append([vertex[20],vertex[23]])
    return UnionVertex, vertex


def EliminaEsterne(obj, aste):
    """
    Cancella gli elementi interamnete esterni alla superficie chiusa
    """
    i=0
    asteToDelete=[]
    for asta in aste:
        s = rs.CurveStartPoint(asta)
        e = rs.CurveEndPoint(asta)
        CBI = rs.CurveBrepIntersect(asta, obj, TrimTol)
        curves = []
        points = []
        if CBI:
            curves, points =CBI
        if (curves == []) and (points == []):
            if not (rs.IsPointInSurface(obj,s,False,TrimTol)) and
not(rs.IsPointInSurface(obj,e,False,TrimTol)) :
```

```
                rs.DeleteObject(asta)
                asteToDelete.append(asta)
        else :
            if points != []:
                rs.DeleteObjects(points)
    for ele in asteToDelete:
        aste.remove(ele)
    return aste


def AccorciaAste(obj, aste):
    """
    Accorcia le aste che fuoriescono dalla superficie chiusa
    """
    asteacc=[]
    pt=[]
    for asta in aste:
        CBI= rs.CurveBrepIntersect(asta, obj, TrimTol)
        if CBI:
            curves, points =CBI
            for p in points: pt.append(p)
            if len(points) == 1:
                s = rs.CurveStartPoint(asta)
                e = rs.CurveEndPoint(asta)
                if rs.IsPointInSurface(obj, s,False,TrimTol) or
rs.IsPointInSurface(obj, e,False,TrimTol):
                    if rs.IsPointInSurface(obj, s):
                        if rs.Distance(points[0], s) >= TrimTol:
                            Astaacc=rs.AddLine(s, points[0])
                            asteacc.append(Astaacc)
                    if rs.IsPointInSurface(obj, e):
                        if rs.Distance(points[0], e) >= TrimTol:
                            Astaacc=rs.AddLine(e, points[0])
                            asteacc.append(Astaacc)
            if len(points)==2:
                Astaacc=rs.AddLine(points[0], points[1])
                asteacc.append(Astaacc)
            elif len(curves)==1:
                asteacc.append(curves[0])
        else:
            asteacc.append(asta)
    return asteacc,pt

#scelgo valori caratteristici
OptCellDim=COD

obj=Shape

#converto la mesh in polysurface se serve
rs.EnableRedraw (False)
if rs.IsMesh(obj):
    mesh = obj
    rs.SelectObject(mesh)
    rs.Command("_MeshToNURB")
    obj = rs.LastCreatedObjects( )
```

```
#ident.ifico la bounding box
vertici = rs.BoundingBox(obj)

#calcolo dimensione box
B1 = rs.Distance(vertici[0],vertici[1])
B2 = rs.Distance(vertici[0] ,vertici[3])
H = rs.Distance(vertici[0],vertici[4])
Volume = B1*B2*H

#calcolo la dimensione delle celle
nxprova = B1/OptCellDim
nyprova = B2/OptCellDim
nzprova = H/OptCellDim

nx = int(nxprova)
ny = int(nyprova)
nz = int(nzprova)

b1 = (B1)/nx
b2 = B2/ny
h = (H)/nz

#vertici di costruzione celle
V = rs.coerce3dpoint([vertici[0][0],vertici[0][1],vertici[0][2]])
punti = []
if CellShape!="4":
    for i in range (nx):
        x = [V.X + i*b1,V.Y,V.Z]
        for j in range (ny):
            y = [V.X + i*b1,V.Y + j *b2,V.Z]
            for u in range (nz):
                pt = [V.X + i*b1,V.Y + j *b2,V.Z + u*h]
                punti.append (pt)
elif CellShape=="4":
    print CellShape
    for i in range (nx+1):
        for j in range (ny):
            for u in range (nz):
                if u%2==0:
                    pt = [V.X + i*b1,V.Y + j *b2,V.Z + u*h]
                elif u%2==1:
                    pt= [V.X+i*b1-b1/2,V.Y + j *b2,V.Z + u*h]
                punti.append (pt)

#inserimento celle
aste=[]
print 'Forma della cella', CellShape
for p in punti :
    if CellShape=="8":
        uv, v = GAMseparata(b1, b2, h, p)
    if CellShape=="7":
        uv, v = GAM(b1, b2, h, p)
    if CellShape=="6":
        uv, v = TSC(b1, b2, h, p)
    if CellShape=="5":
        uv, v = OctectTruss(b1, b2, h, p)
```

```
    if CellShape=="4":
        uv, v = WallachGibson2001(b1, b2, h, p)
    if CellShape=="3":
        uv, v = ReinforcedBodyCenterCubic(b1, b2, h, p)
    if CellShape=="2":
        uv, v = BodyCenterCubic(b1, b2, h, p)
    if CellShape=="1":
        uv, v = SC.CubicCell(b1, b2, h, p)
    for linea in uv:
        asta=rs.AddLine(linea[0],linea[1])
        aste.append(asta)

print "aste prima di eliminazione esterne",len(aste)
aste=EliminaEsterne(obj, aste)
print "aste prima di accorciare",len(aste)

aste,pt=AccorciaAste(obj,aste)
print "aste finali", len(aste)

l = 0
for linea in aste:
    l = l+rs.CurveLength(linea)
```

Ottimizzazione B

```
import math as mt

def MaxSco(Ut,Rad):
    Rn=[]
    Op=-1
    Opmin=1
    Fobj=0
    ins=0
    NVG=0
    for i,R in enumerate(Rad):
        Rni=R*mt.sqrt(Ut[i]/UtiC)  #sqrt
        if Rni<Rmin:
            Rni=Rmin
        elif Rni>=Rmax:
            Rni=Rmax
        Rn.append(Rni)
        if Ut[i]>((1+0.1)*UtiT):
            NVG=NVG+1
            if R==Rmax:
                ins=ins+1
        op=Ut[i]-UtiT
        if op>Op:
            Op=op
        elif op<Opmin and Rni!=Rmin:
            Opmin=op
    if abs(Op)>=abs(Opmin):
        Fobj=abs(Op/UtiT)*100
    else:
        Fobj=abs(Opmin/UtiT)*100
    Mass=UtiT+Op
```

```
    Min=UtiT+Opmin
    return(ins,NVG,Mass,Min,Fobj)

def MaxDR(Ut,Rad):
    Rn=[]
    DR=0
    Op=-1
    Opmin=1
    Fobj=0
    ins=0
    NVG=0
    for i,R in enumerate(Rad):
        Rni=R*mt.sqrt(Ut[i]/UtiC)  #sqrt
        if Rni<Rmin:
            Rni=Rmin
        elif Rni>=Rmax:
            Rni=Rmax
        Rn.append(Rni)
        if Ut[i]>((1+0.1)*UtiT):
            NVG=NVG+1
            if R==Rmax:
                ins=ins+1
        op=Ut[i]-UtiT
        if op>Op:
            Op=op
        elif op<Opmin and Rni!=Rmin:
            Opmin=op
        dr=abs(Rni-R)/R*100
        if dr>=DR:
            DR=dr
    Fobj=DR
    Mass=UtiT+Op
    Min=UtiT+Opmin
    return(ins,NVG,Mass,Min,Fobj)

def DistSco(Ut,Rad):
    Rn=[]
    Op=-1
    Opmin=1
    Fobj=0
    ins=0
    NVG=0
    for i,R in enumerate(Rad):
        Rni=R*mt.sqrt(Ut[i]/UtiC)  #sqrt
        if Rni<Rmin:
            Rni=Rmin
        elif Rni>=Rmax:
            Rni=Rmax
        Rn.append(Rni)
        if Ut[i]>((1+0.1)*UtiT):
            NVG=NVG+1
            if R==Rmax:
                ins=ins+1
        op=Ut[i]-UtiT
        if op>Op:
            Op=op
```

```
            elif op<Opmin and Rni!=Rmin:
                Opmin=op
        Mass=UtiT+Op
        Min=UtiT+Opmin
        Fobj=(Mass-Min)/UtiT*100
        return(ins,NVG,Mass,Min,Fobj)

    def SumDR(Ut,Rad):
        Rn=[]
        DR=0
        Op=-1
        Opmin=1
        Fobj=0
        ins=0
        NVG=0
        for i,R in enumerate(Rad):
            Rni=R*mt.sqrt(Ut[i]/UtiC)  #sqrt
            if Rni<Rmin:
                Rni=Rmin
            elif Rni>=Rmax:
                Rni=Rmax
            Rn.append(Rni)
            if Ut[i]>((1+0.1)*UtiT):
                NVG=NVG+1
                if R==Rmax:
                    ins=ins+1
            op=Ut[i]-UtiT
            if op>Op:
                Op=op
            elif op<Opmin and Rni!=Rmin:
                Opmin=op
            dr=abs(Rni-R)/R
            Fobj=Fobj+dr
        Mass=UtiT+Op
        Min=UtiT+Opmin
        return(ins,NVG,Mass,Min,Fobj)


    if scelta==1:
        ins,NVG,Mass,Min,Fobj=MaxSco(Ut,Rad)
    if scelta==2:
        ins,NVG,Mass,Min,Fobj=MaxDR(Ut,Rad)
    if scelta==3:
        ins,NVG,Mass,Min,Fobj=DistSco(Ut,Rad)
    if scelta==4:
        ins,NVG,Mass,Min,Fobj=SumDR(Ut,Rad)

    if togg==False:
        if Fobj<=obj or ins>0 :
            flag=False
        else:
            flag=True
    else:
        if I<nit:
            flag=True
        else:
```

```
        flag=False

print "Eseguite ",I," iterazioni"
print "Aste insufficienti: ",ins
print "Aste non verificate G: ",NVG
print "Utilizzazione massima",Mass
print "Utilizzazione minima",Min
if scelta!=4:
    print "Funzione obbiettivo",round(Fobj,2),"%"
elif scelta==4:
    print "Funzione obbiettivo",round(Fobj,2)
```

Ottimizzazione B&S

```
import math as mt
Fobj=0
FobjS=0
Mass=0
Min=0
SMass=0
SMin=0
Op=-1
Opmin=1
SOpmin=1
RNVG=0
SNVG=0
Rins=0
Sins=0
Rads=[]
Spes=[]

if I<nitB:
    for i,R in enumerate(Rad[0:Lng]):
        Rni=R*mt.sqrt(Ut[i]/UtBT)
        if Rni>=Rmax:
            Rni=Rmax
        elif Rni<=Rmin:
            Rni=Rmin
        Rads.append(Rni)
        if Ut[i]>=(1.1)*UtBT:
            RNVG=RNVG+1
            if R==Rmax:
                Rins=Rins+1
        op=Ut[i]-UtBT
        if op>Op:
            Op=op
        elif op<=Opmin and R!=Rmin:
            Opmin=op
    Mass=UtBT+Op
    Min=UtBT+Opmin
    if abs(Op)>=abs(Opmin):
        Fobj=abs(Op/UtBT)*100
    else:
        Fobj=abs(Opmin/UtBT)*100
    for j in Rad[Lng:len(Rad)]:
```

```
        Spes.append(Smin)

elif I==nitB:
    for j,S in enumerate(RadF[0:Lng]):
        Rads.append(S)
        if Ut[j]>=(1.1)*UtBT:
            RNVG=RNVG+1
            if S==Rmax:
                Rins=Rins+1
        op=Ut[j]-UtBT
        if op>Op:
            Op=op
        elif op<=Opmin and S!=Rmin:
            Opmin=op
    Mass=UtBT+Op
    Min=UtBT+Opmin
    if abs(Op)>=abs(Opmin):
        Fobj=abs(Op/UtBT)*100
    else:
        Fobj=abs(Opmin/UtBT)*100
    for k,T in enumerate(Rad[Lng:len(Rad)]):
        Spes.append(Smax)
        sop=Ut[k+Lng]-UtST
        if sop<SOpmin:
            SOpmin=sop
    SMass= max(Ut[Lng:len(Rad)])
    SMin=UtST+SOpmin

elif I>nitB:
    for j,S in enumerate(RadF[0:Lng]):
        Rads.append(S)
        if Ut[j]>=(1.1)*UtBT:
            RNVG=RNVG+1
            if S==Rmax:
                Rins=Rins+1
        op=Ut[j]-UtBT
        if op>Op:
            Op=op
        elif op<=Opmin and S!=Rmin:
            Opmin=op
    Mass=UtBT+Op
    Min=UtBT+Opmin
    if abs(Op)>=abs(Opmin):
        Fobj=abs(Op/UtBT)*100
    else:
        Fobj=abs(Opmin/UtBT)*100
    for k,T in enumerate(Rad[Lng:len(Rad)]):
        if Ut[k+Lng]>UtST:
            Tni=T*(4*(Ut[k+Lng]/UtST)**2-8*(Ut[k+Lng]/UtST)+5)
        else:
            Tni=T*(0.5+0.5*mt.sqrt(1-((Ut[k+Lng]-UtST)/UtST)**2))
#ellissi
        if Tni>Smax:
            Tni=Smax
        elif Tni<Smin:
            Tni=Smin
```

```
        Spes.append(Tni)
        if Ut[k+Lng]>=1.1*UtST:
            SNVG=SNVG+1
            if T==Smax:
                Sins=Sins+1
        sop=Ut[k+Lng]-UtST
        if sop<SOpmin and T!=Smin:
            SOpmin=sop
    SMass= max(Ut[Lng:len(Rad)])
    SMin=UtST+SOpmin
    SOp=SMass-UtST
    if abs(SOp)>=abs(SOpmin):
        FobjS=abs(SOp/UtBT)*100
    else:
        FobjS=abs(SOpmin/UtBT)*100


NewRad=Rads+Spes

if I>=nitB or Fobj<=obj or Rins>0:
    flag=False
else:
    flag=True

print "Eseguite ",I," iterazioni"
print "Aste insufficienti: ",Rins
print "Aste non verificate G: ",RNVG
print "Utilizzazione massima B",Mass
print "Utilizzazione minima B",Min
print "Funzione obbiettivo B",round(Fobj,2),"%"
print "Utilizzazione massima S",SMass
print "Utilizzazione minima S",SMin
print "Funzione obbiettivo S",round(FobjS,2),"%"
```

Ottimizzazione G

```
import math as mt
Optimize=0
op=0
# definizione del vettore k da riempire con i valori
# di op
k=[]
for i,R in enumerate(Rad):
    op=((UtiT-Uti[i])/UtiT)**2
# elementi non caricati
    if (Uti[i]<0.0001):
        op=((R-LS)/LS)**2
# penalizzazione per gli elementi con Uti<UtiT
    if (Uti[i]>UtiT):
        op=100*op
    k.append(op)
    Optimize=(Optimize+op)
Opt=mt.sqrt(Optimize/len(Rad))
```

Bibilografia.

[1] C. Chu, G. Graft, D.W. Rosen: Design for additive manufacturing of cellular structures, Computer-Aided Design & Applications, 5(5), 2008, pages 686 696.

[2] V.S. Deshpande, N.A. Fleck, M.F. Ashby: Effective properties of the octect-truss lattice material, journal of the mechanics and physics of solids, volume 49, issue 2, 22 january 2001, pages 1747-1769

[3] J. Stampfl, M.M. Seyr, M.H. Luxner, H.E. Pettermann, A.Woesz, P.Fratzl: Regular low density cellular structures – rapid prototyping, numerical simulation, mechanical testing, MRS Proceedings 2004

[4] A.P. Roberts, E.J. Garboczi: Elastic Properties of model random three-dimensional open cell solids journal of the mechanics and physics of solids, volume 50, issue 1, January 2002, pages 33-55.

[5] J.L. Gibson: Biomechanics of cellular solids journal of biomechanics, volume 38, issue 3 march 2005, pages 377-399.

[6] J.C. Wallach, L.J.Gibson: Difect Sensitivity of a 3d truss material, scipta materiala 45,2001, pages 639-644.

[7] D. Goldberg: Genetic algorithms in search, optimization and machine learning, Addison-Wesley Professional, Reading (MA), 1989.