Reduced order models and machine learning in analysis and optimum design of structures

Nikolaos Ath. Kallioras

Department of Structural Engineering
School of Civil Engineering
National Technical University of Athens

This dissertation is submitted for the degree of
Doctor of Philosophy

School of Civil Engineering April 2019

Committee

1. N. D. Lagaros
   Associate Professor, Department of Structural Engineering,
   School of Civil Engineering, Supervisor .....

2. K. Giannakoglou
   Professor, Lab. of Thermal Turbomachines,
   School of Mechanical Engineering, Member of the Committee .....

3. Vl. Koumousis
   Professor, Department of Structural Engineering,
   School of Civil Engineering .....

4. Ch. Provatidis
   Professor, Department of Mechanical Design & Control Section,
   School of Mechanical Engineering .....

5. E. Plevris
   Associate Professor, Department of Civil Engineering & Energy
   Technology, Oslo Metropolitan University .....

6. D. Charmpis
   Associate Professor, Department of Civil & Environmental
   Engineering, University of Cyprus .....

7. G. Tsiatas
   Associate Professor, Department of Mathematics
   University of Patras .....

Εκτενής Περίληψη Διδακτορικής Διατριβής

1. Γενική Εισαγωγή

Ο βέλτιστος σχεδιασμός κατασκευών αποτέλεσε αντικείμενο έρευνας, πρακτικά, από την περίοδο που πραγματοποιήθηκαν οι πρώτες ανθρώπινες κατασκευές. Η ανάγκη για την εύρεση του βέλτιστου σχεδιασμού συνέχισε να αυξάνει με την πάροδο του χρόνου, με την αύξηση να μπορεί να χαρακτηριστεί ως γεωμετρική καθώς αυξάνονταν και τα μεγέθη αλλά και η πολυπλοκότητα των κατασκευών. Η ανάλυση των κατασκευών και ειδικότερα το υπολογιστικό κόστος αυτών, ως ανάλογο του μεγέθους και της πολυπλοκότητας των κατασκευών, παρουσίασε και αυτό αντίστοιχη αύξηση. Το γεγονός αυτό συνετέλεσε στην στροφή μεριδίου του ενδιαφέροντος της έρευνας σε προσεγγιστικές μεθόδους, καθώς οι μέθοδοι ακριβούς υπολογισμού γίνονταν ιδιαίτερα απαιτητικές. Αποτελεί επίσης αξιοσημείωτο γεγονός πως και οι διαθέσιμοι υπολογιστικοί πόροι αυξήθηκαν σημαντικά τα τελευταία χρόνια, ιδιαίτερα με την χρήση των επιταχυντών.

Ενδεικτικά, οι προσεγγιστικές μέθοδοι που χρησιμοποιούνται ευρέως στη διεθνή βιβλιογραφία είναι τα μοντέλα μειωμένης τάξης, τα τεχνητά νευρωνικά δίκτυα αλλά και οι gradient-free αλγόριθμοι βελτιστοποίησης. Κάθε μία από τις μεθόδους αυτές παρουσιάζει ξεχωριστά χαρακτηριστικά, προτερήματα αλλά και μειονεκτήματα. Για παράδειγμα, η συμπεριφορά των gradient-free αλγόριθμων βελτιστοποίησης εξαρτάται από το πλήθος των παραμέτρων κάθε προβλήματος, ενώ τα νευρωνικά δίκτυα είχαν παραγκωνιστεί όσο ήταν αδύνατη η εκπαίδευση βαθιών αρχιτεκτονικών. Στον αντίποδα, τα τελευταία χρόνια μέσω της επιτυχούς εκπαίδευσης βαθιών νευρωνικών δικτύων, έχει σημειωθεί εξαιρετική αύξηση του ερευνητικού ενδιαφέροντος γύρω από αυτά σε τομείς όπως natural language processing, computer vision και big data. Επιπλέον, τα μοντέλα μειωμένης τάξης χρησιμοποιούνται ευρέως σε προβλήματα ανάλυσης πολύπλοκων και μεγάλων μοντέλων.

Η συμβολή της παρούσας διατριβής επικεντρώνεται στον τομέα της ανάλυσης και του βέλτιστου σχεδιασμού δομικών κατασκευών μέσω της ανάπτυξης νέων υπολογιστικών μεθόδων συνδυάζοντας ακριβείς και προσεγγιστικές μεθόδους. Η ροπή προς την εκμετάλλευση μεθόδων soft computing είναι αναπόφευκτη, κατά την κρίση του συγγραφέα, λόγω της δυσαναλογίας που παρουσιάζεται στην αύξηση του μεγέθους και της πολυπλοκότητας των σημερινών προβλημάτων που καλείται να αντιμετωπίσει ο μηχανικός και στην αύξηση της διαθέσιμης υπολογιστικής ισχύος.

Οι ακριβείς μέθοδοι έχουν την ικανότητα να μειώνουν δραστικά την τιμή της αντικειμενικής συνάρτησης του προβλήματος μετά από ένα σχετικά μικρό πλήθος

επαναλήψεων. Δυστυχώς, όμως, δεν έχουν τη δυνατότητα να αποφύγουν τον εγκλωβισμό τους σε κάποιο τοπικό ελάχιστο. Πιο συγκεκριμένα, σε προβλήματα πραγματικής φύσης και κλίμακας, είναι πρακτικά σίγουρο πως η λύση που θα προτείνει ένας αλγόριθμος αυτής της οικογένειας θα είναι ένα τοπικό βέλτιστο και όχι το ολικό. Επιπλέον, αξίζει να σημειωθεί πως ο υπολογισμός της παραγώγου πρώτης και δεύτερης τάξης που χρησιμοποιούν οι ακριβείς μέθοδοι, απαιτούν σημαντικό υπολογιστικό φόρτο ενώ σε αρκετές περιπτώσεις είναι και αδύνατο να υπολογιστούν. Οι παραπάνω λόγοι συνεισφέρουν στην ανάπτυξη και χρήση μεθόδων soft computing σε πληθώρα προβλημάτων πραγματικής φύσης.

Παρουσιάζοντας μια σύντομη ιστορική αναδρομή στην εξέλιξη και χρήση των μεθόδων soft computing, αξίζει να σημειωθεί η σημαντική ερευνητική προσοχή που δέχθηκαν στα χρόνια μεταξύ 1970 και 1990, όπου είναι γνωστή και η χρήση τους σε πολλά προβλήματα πραγματικής κλίμακας. Οι gradient free μέθοδοι, λόγω της τυχαιότητας που περιλαμβάνουν στη δομή τους έχουν τη δυνατότητα να ξεπερνούν τυχόν εγκλωβισμούς σε τοπικά ελάχιστα και, πρακτικά, αν δεν έχουν περιορισμό τερματισμού, εγγυώνται την εύρεση του ολικού ελάχιστου. Βασικό μειονέκτημα αυτών των μεθόδων είναι η ταχύτητα σύγκλισής τους καθώς απαιτούν σημαντικό πλήθος επαναληπτικών επιλύσεων. Το τελευταίο οδήγησε και στην μείωση του ενδιαφέροντος για αυτές τις τεχνικές με την αύξηση της πολυπλοκότητας των προβλημάτων.

Την τελευταία δεκαετία, συγκεντρώθηκε μεγάλο ενδιαφέρον γύρω από τις μεθόδους soft computing κυρίως λόγω του μεγάλου όγκου δεδομένων που συλλέγουν και πρέπει να διαχειριστούν οι εταιρίες online υπηρεσιών. Το ενδιαφέρον αυτό οδήγησε στην σύλληψη και παρουσίαση νέων μεθόδων, ικανών να διαχειριστούν προβλήματα αυξημένης πολυπλοκότητας όπως, για παράδειγμα, τα βαθιά νευρωνικά δίκτυα που πλέον έχουν γίνει γνωστά σε διάφορες πτυχές τόσο της επιστημονικής έρευνας όσο και της αγοράς. Στην παρούσα διατριβή προτείνονται διάφορες μέθοδοι χρήσης σύγχρονων τεχνικών soft computing στην επιστήμη του πολιτικού μηχανικού αλλά και ειδικότερα στην ανάλυση και τον βέλτιστο σχεδιασμό κατασκευών. Ο υπολογιστικός φόρτος των περισσότερων προβλημάτων στην ανάλυση και τον σχεδιασμό κατασκευών αφορά την επαναληπτική επίλυση της παρακάτω εξίσωσης:

$$\{P\} = [K] * \{U\} \tag{1.1}$$

Αυτό οφείλεται στο κόστος υπολογισμού του αντίστροφου μητρώου, του μητρώου δυσκαμψίας που απαιτείται για την επίλυση της εξίσωσης. Για την αντιμετώπιση του παραπάνω προβλήματος έχουν παρουσιαστεί διάφορες τεχνικές που σχετίζονται με τη μείωση του μεγέθους του μητρώου δυσκαμψίας, τη μείωση του απαιτούμενου αριθμού επίλυσης της εξίσωσης ισορροπίας, την χρήση τεχνικών παράλληλης επεξεργασίας αλλά και συνδυασμούς των όλων ή μερικών από τις τεχνικές αυτές. Στην παρούσα διδακτορική διατριβή παρουσιάζονται νέες μέθοδοι που αφορούν τις τεχνικές αυτές, αλλά και μία μέθοδος που δύναται να χρησιμοποιηθεί στον τομέα του Generative Design.

Η παρούσα διατριβή είναι χωρισμένη σε επτά κεφάλαια. Στο 1ο κεφάλαιο συναντάται μία εισαγωγή και σύντομη ανάλυση των περιεχομένων της. Στο 2ο κεφάλαιο παρουσιάζεται η γενική ιδέα της μαθηματικής διατύπωσης των προβλημάτων βελτιστοποίησης, μία βιβλιογραφική ανασκόπηση των μεθόδων που χρησιμοποιούνται στην βελτιστοποίηση αλλά και αναλυτικότερες περιγραφές αλγορίθμων που έχουν χρησιμοποιηθεί στα πλαίσια της διατριβής. Στη συνέχεια του κεφαλαίου αυτού παρουσιάζονται εκτενώς μία βελτιωμένη εκδοχή ενός υπάρχοντος μεταευρετικού αλγορίθμου αλλά και ένας νέος προτεινόμενος μεταευρετικός αλγόριθμος. Στο 3ο κεφάλαιο παρουσιάζονται μια βιβλιογραφική ανασκόπηση των βαθιών νευρωνικών δικτύων αλλά και αναλυτική παρουσίαση των τύπων δικτύων που χρησιμοποιήθηκαν στο πλαίσιο της παρούσας διδακτορικής διατριβής. Στο 4ο κεφάλαιο παρουσιάζεται μια μέθοδος μείωσης του υπολογιστικού φόρτου της βελτιστοποίησης τοπολογίας που βασίζεται σε βαθιά νευρωνικά δίκτυα και αναπτύχθηκε στο πλαίσιο της διδακτορικής διατριβής. Στο 5ο κεφάλαιο παρουσιάζονται τρεις μέθοδοι παραγωγής μοντέλων μειωμένης τάξης μέσω βαθιών νευρωνικών δικτύων και η χρήση τους στη βελτιστοποίηση τοπολογίας. Στο 6ο κεφάλαιο παρουσιάζεται μια μέθοδος που χρησιμοποιεί την βελτιστοποίηση τοπολογίας και βαθιά νευρωνικά δίκτυα στον τομέα του Generative Design. Τέλος, στο 7ο κεφάλαιο παρουσιάζονται κάποιες προτάσεις για μελλοντική έρευνα στους παραπάνω τομείς.

2. Βελτιστοποίηση και απόδοση αλγορίθμων

Ως βελτιστοποίηση, περιγράφεται η βελτίωση μιας συγκεκριμένης λύσης ως προς προκαθορισμένα κριτήρια και κάτω από συγκεκριμένους περιορισμούς. Στο αντικείμενο του δομοστατικού μηχανικού η βελτιστοποίηση αποσκοπεί στην εύρεση του βέλτιστου σχήματος, μεγέθους ή της βέλτιστης τοπολογίας ενός δομικού συστήματος ως προς το κόστος ή την απόδοση αυτού χωρίς την παραβίαση καθορισμένων περιορισμών. Οι μέθοδοι που χρησιμοποιούνται μπορούν να κατηγοριοποιηθούν με διάφορους τρόπους. Οι πιο διαδεδομένες κατηγοριοποιήσεις είναι:

- Ντετερμινιστικές ή Στοχαστικές μέθοδοι.
- Μέθοδοι Gradient-based ή Gradient-free.
- Μέθοδοι Τοπικής ή Καθολικής αναζήτησης (Local Search, Global Search).

Η γενική διατύπωση ενός προβλήματος βελτιστοποίησης χωρίς περιορισμούς μπορεί να περιγραφεί ως εξής:

$$\begin{aligned} &Minimize \quad F(X) \quad where: \\ &\qquad X = [x_1, x_2, ..., x_{n-1}, x_n] \\ &with \quad respect \quad to: \\ &\qquad x_i \in [x_i^{L_i}, x_i^{U_i}] \end{aligned} \tag{1.2}$$

ενώ στην περίπτωση του προβλήματος βελτιστοποίησης με περιορισμούς περιγράφεται ως:

$$
\begin{aligned}
&Minimize \quad F(X) \quad where: \\
&\qquad X = [x_1, x_2, ..., x_{n-1}, x_n] \\
&with \quad respect \quad to: \\
&\qquad g_k(X) \le 0, \qquad k = 1, 2, ..., m-1, m \\
&\qquad l_j(X) = 0, \qquad j = 1, 2, ..., s-1, s \\
&\qquad x_i \in [x_i^{L_i}, x_i^{U_i}]
\end{aligned}
\tag{1.3}
$$

όπου F(X) είναι η αντικειμενική συνάρτηση που θα βελτιστοποιηθεί, X είναι το διάνυσμα λύσης, g(X) οι ανισοτικοί περιορισμοί, l(X) οι ισοτικοί περιορισμοί και $x_i^{L_i}$, $x_i^{U_i}$ τα όρια της x_i μεταβλητής.

Οι Gradient-based αλγόριθμοι βασίζονται στην πληροφορία παραγώγου 1ης και συνήθως 2ης τάξης για την εύρεση της βέλτιστης λύσης. Μερικές από τις πιο γνωστές μεθόδους είναι οι:

- Steepest descent algorithm [22]
- Conjugate gradient method [67]
- Newton-Raphson method [15]
- Quasi-Newton method [19,37]

Επιπλέον, δύο αλγόριθμοι που χρησιμοποιήθηκαν στο πλαίσιο της παρούσας διδακτορικής διατριβής είναι οι:

- Optimality criteria algorithm [31]
- Method of moving asymptotes [202]

Οι Gradient-free αλγόριθμοι δεν πραγματοποιούν υπολογισμό παραγώγου και χρησιμοποιούνται σε περιπτώσεις που δεν είναι δυνατόν να γίνει υπολογισμός της παραγώγου ή αυτό 'κοστίζει' υπερβολικά ή δεν είναι ακριβής ο υπολογισμός [173]. Ξεκινώντας από την μέθοδο SIMPLEX [150] και μέχρι σήμερα υπάρχουν πολλές εφαρμογές σε διάφορα προβλήματα από δομοστατικής φύσης [109,113,114] μέχρι ιατρικής φύσης [136].

2.1 Harmony Search και Improved Harmony Search

Ένας από τους ευρέως διαδεδομένους μεταευρετικούς αλγόριθμους είναι ο Harmony Search (HS) [60] που δημιουργήθηκε από την μοντελοποίηση του αυτοσχεδιασμού των μουσικών μιας Jazz ορχήστρας. Σύμφωνα με τη διαδικασία αυτή, κάποιο μέλος της ορχήστρας ξεκινά παίζοντας μία νότα και κάποιο άλλο μέλος της μπάντας αποφασίζει να παίξει μια νέα νότα είτε επιλέγοντάς την τυχαία είτε επιλέγοντάς την από τα ακούσματα-μνήμη του είτε επιλέγοντας κάποια με βάση τα ακούσματά του και παραλλάσσοντάς την λίγο. Για την μαθηματική μοντελοποίηση του αλγόριθμου, καλό είναι να οριστούν κάποιες αντιστοιχίες: οι μουσικοί

αντιστοιχούν σε μεταβλητές σχεδιασμού, το εύρος κάθε οργάνου στο εύρος κάθε μεταβλητής, η παραγόμενη μελωδία αντιστοιχεί σε διάνυσμα λύσης και η αποδοχή της μελωδίας από το κοινό στην ποιότητα της λύσης.

Οι βασικές λειτουργίες του HS είναι α) Αρχικοποίηση μνήμης, β) Παραγωγή νέας μελωδίας και γ) Ενημέρωση μνήμης. Κατά το πρώτο βήμα, γεμίζει η μνήμη με τυχαία παραγόμενες λύσεις. Στο δεύτερο βήμα παράγεται μια νέα μελωδία επιλέγοντας κάθε μέλος του διανύσματος με μία από τις εξής επιλογές: α) Τυχαία, β) Επιλέγοντας μία από τη μνήμη, γ) Επιλέγοντας μία από τη μνήμη και αλλάζοντάς την λίγο. Τέλος, στο τρίτο βήμα ελέγχεται, αν η τιμή της αντικειμενικής συνάρτησης που αντιστοιχεί στο παραχθέν διάνυσμα λύσης είναι καλύτερη από την χειρότερη από όσες είναι αποθηκευμένες στη μνήμη και αν ναι, τότε την αντικαθιστά. Διαφορετικά, εγκαταλείπεται. Τα παραπάνω μπορούν να αποτυπωθούν ως εξής:

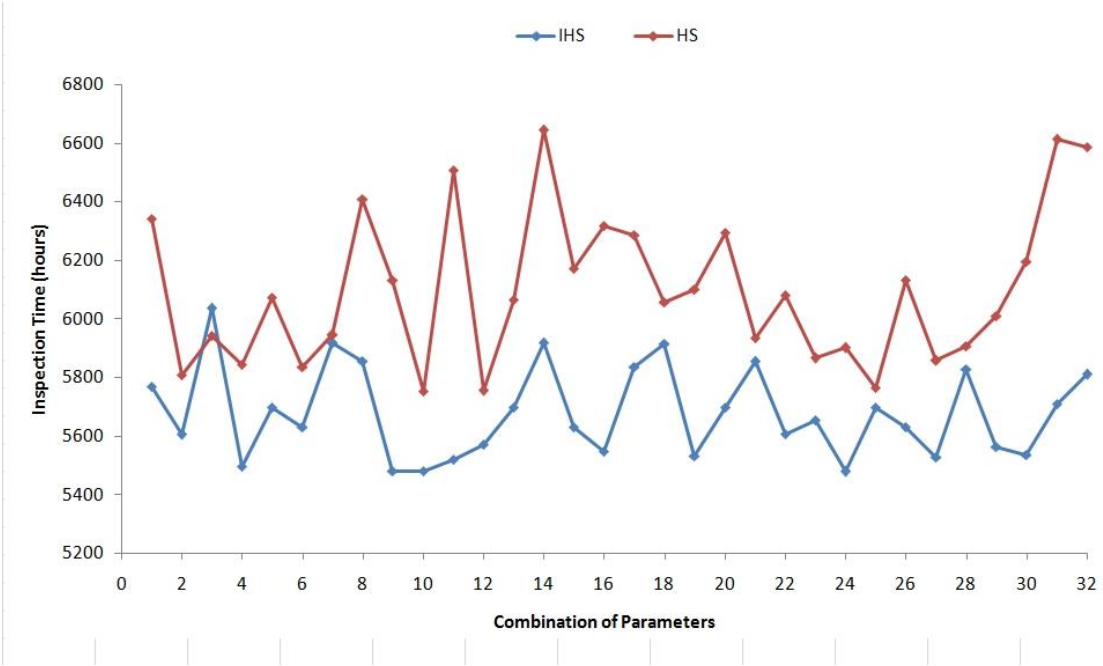
$$s_i^{New} = \begin{cases} s_i \in [x_i^{Lower}, x_i^{Upper}] & \text{with probability 1-HMCR} \\ s_i \in HM = [s_i^1, s_i^2, \cdots, s_i^{HMS}] & \text{with probability HMCR*(1-PAR)} \\ s_i + k & \text{with probability HMCR*PAR} \end{cases} \quad (1.4)$$

όπου s_i είναι το I στοιχείο του διανύσματος λύσης s, HM η μνήμη του αλγόριθμου, HMCR και PAR είναι παράμετροι αυτού και x_i^{Lower}, x_i^{Upper} τα άνω και κάτω όρια της I μεταβλητής σχεδιασμού.

Στα πλαίσια της παρούσας διδακτορικής διατριβής διαμορφώθηκε μια νέα, βελτιωμένη εκδοχή του HS που ονομάζεται Improved Harmony Search (IHS). Οι δύο βασικές αλλαγές αφορούν: α) την κατάργηση της επιλογής τιμής από τη μνήμη χωρίς αλλαγή της και β) την παραγωγή ολόκληρου του διανύσματος λύσης από μία από τις δύο, πλέον, λειτουργίες και όχι κάθε μεταβλητής αυτής ξεχωριστά. Αντίστοιχα, η εξίσωση που περιγράφει τις νέες λειτουργίες είναι:

$$s^{New} = \begin{cases} \forall\, i \in [1,n] : s_i \in [x_i^{Lower}, x_i^{Upper}] & \text{with probability 1-HMCR} \\ \forall\, i \in [1,n] : s_i \in HM = [s_i^1, s_i^2, \cdots, s_i^{HMS}] & \text{with probability HMCR} \end{cases} \quad (1.5)$$

Οι παραπάνω αλλαγές πραγματοποιήθηκαν στοχεύοντας στη βελτίωση της συμπεριφοράς του αλγορίθμου τόσο από άποψη σταθερότητας όσο και από απόδοσης ως προς την εύρεση του βέλτιστου. Τα παραπάνω επιβεβαιώθηκαν και μέσω εφαρμογής του HS και του IHS σε πραγματικά προβλήματα βελτιστοποίησης. Ενδεικτικά, αναφέρεται η βέλτιστη διαμέριση της πόλης της Θεσσαλονίκης σε περιοχές ευθύνης συνεργείων επιθεώρησης, ώστε σε περίπτωση σεισμικού συμβάντος να ολοκληρωθεί η επιθεώρηση όσο το δυνατόν ταχύτερα. Εξετάστηκε τόσο η απόδοση των αλγόριθμων (μικρότερος αναγκαίος χρόνος για την επιθεώρηση) όσο και η σταθερότητά τους καθώς πραγματοποιήθηκαν 32 διαφορετικές επιλύσεις με 32 διαφορετικά σετ παραμέτρων. Τα αποτελέσματα φαίνονται στην παρακάτω εικόνα και τον παρακάτω πίνακα:



	IHS	HS
Minimum Time (h)	5478.61	5751.64
Maximum Time (h)	6034.62	6643.47
Average Time (h)	5678.24	6097.35
Standard Deviation (h)	154.29	260.34
COV (%)	2.72	4.27

Από την εικόνα είναι φανερό πως ο IHS συμπεριφέρεται καλύτερα από τον HS ανεξαρτήτως παραμέτρων.

2.2 Pity Beetle Algorithm

Στα πλαίσια της παρούσας διδακτορικής διατριβής αναπτύχθηκε ένας νέος μεταευρετικός αλγόριθμος με το όνομα Pity Beetle Algorithm (PBA). Ο αλγόριθμος αυτός ανήκει στο είδος των εμπνευσμένων από τη φύση αλγόριθμων και στον τύπο των αλγορίθμων πλήθους (swarm). Για την ακρίβεια, είναι εμπνευσμένος από τις φυσικές διεργασίες ενός είδους σκαθαριού που ονομάζεται Pityogenes chalcographus και συναντάται στα δάση της Κεντρικής και Βόρειας Ευρώπης αλλά και στην Ελλάδα [107,9]. Ο PBA έχει βασιστεί στην χαρακτηριστική συμπεριφορά που το είδος αυτό επιδεικνύει κατά την αναζήτηση τροφής και κατάλληλων δέντρων ώστε να δημιουργεί φωλιές, τρεφόμενο από τον φλοιό των κορμών των δέντρων και δημιουργώντας φωλιές στο εσωτερικό του κορμού.

Το είδος αυτό χαρακτηρίζεται από την ικανότητά του να εποικεί γρήγορα εντός ενός δάσους και να το καλύπτει ξεκινώντας από μια μικρή αποικία [186]. Συγκεκριμένα, μια μικρή ομάδα σκαθαριών πετάνε τυχαία εντός του δάσους σε αναζήτηση μη υγιών δέντρων, ώστε να σχηματίσουν εκεί τις φωλιές τους. Όταν βρεθεί αυτό, ελευθερώνουν μια φερομόνη ελκύοντας και άλλα άτομα να κινηθούν προς το δέντρο αυτό. Αν αντίστοιχα έχει δημιουργηθεί υπερπληθυσμός σε μία θέση, τότε εκλύει μια άλλη φερομόνη που αποτρέπει να κινηθούν άλλα άτομα προς την περιοχή αυτή.

Κατά συνέπεια, γίνεται η δημιουργία αποικίας χτίζοντας φωλιές και προσελκύοντας θηλυκά άτομα. Επιδεικνύοντας πολυγαμική συμπεριφορά, κάθε αρσενικό θα δημιουργήσει 3 ως 6 νέες γενιές με διαφορετικά θηλυκά άτομα με κάθε μία από αυτές να αποτελείται από 70 περίπου νέα άτομα. Οι νέες γενιές θα κινηθούν προς αναζήτηση νέων μη υγιών δέντρων για να δημιουργήσουν τις δικές τους αποικίες. Η απόσταση αναζήτησης που μπορούν να καλύψουν εξαρτάται από την ποιότητα τροφής της αρχικής τους θέσης. Είναι κατανοητό πως μπορούν εύκολα να αυξήσουν τον πληθυσμό τους σχετικά γρήγορα και όταν υπάρχει ικανός πληθυσμός έχουν τη δυνατότητα να επιτεθούν και σε υγιή δέντρα.

Ο αλγόριθμος PBA αποτελείται από τρεις κύριες λειτουργίες: α) Αρχικοποίηση, β) Μέθοδος εύρεσης νέας θέσης και γ) Ενημέρωση θέσεων αποικιών. Αρχικά, δημιουργείται η πρώτη γενιά με την ομάδα ατόμων που μετακινούνται τυχαία στο δάσος – πεδίο ορισμού για την εύρεση της θέσης της πρώτης αποικίας. Όταν αυτή εντοπιστεί δημιουργείται η πρώτη αποικία και κάθε νέα γενιά που θα δημιουργηθεί ψάχνει για νέες θέσεις με βάση συγκεκριμένους τρόπους μετακίνησης. Όταν βρεθούν οι νέες θέσεις, ενημερώνεται το αρχείο θέσεων αποικιών. Αυτή η διαδικασία συνεχίζεται ως ότου ικανοποιηθεί το κριτήριο τερματισμού του αλγόριθμου.

Η δημιουργία του υπερκύβου των πιθανών διακριτών θέσεων εντός της περιοχής μετακίνησης γίνεται με τη χρήση μιας Random Sampling τεχνικής (RST) [93] παρόμοια με την Latin Hypercube Sampling [152]. Η αρχική τυχαία τοποθέτηση των ατόμων μπορεί να περιγραφεί ως εξής:

$$\begin{aligned} & x_j^{(0)} = RST(L, U, D, N_{pop}) \\ & \text{where} \\ & L = [L_1, L_2, \ldots, L_D], U = [U_1, U_2, \ldots, U_D] \text{ and } j = 1, 2, \ldots, N_{pop} \end{aligned} \tag{1.6}$$

όπου j είναι το άτομο της (0) γενιάς, x η θέση του, N_{pop} το πλήθος των αρχικών ατόμων και L_i, D_i τα ελάχιστα και μέγιστα όρια της i μεταβλητής σχεδιασμού.

Οι τρόποι μετακίνησης των ατόμων στον PBA είναι 5:

- Μικρού εύρους μετακίνηση
- Μεσαίου εύρους μετακίνηση
- Μεγάλου εύρους μετακίνηση
- Μετακίνηση με μέγιστο εύρος
- Μετακίνηση με εκμετάλλευση της μνήμης

Οι πρώτες τέσσερις μετακινήσεις είναι ταυτόσημες λειτουργικά και διαφέρουν ως προς το εύρος ενώ η τελευταία είναι διαφορετική ως προς την λειτουργία. Το εύρος ορίζεται ως εξής:

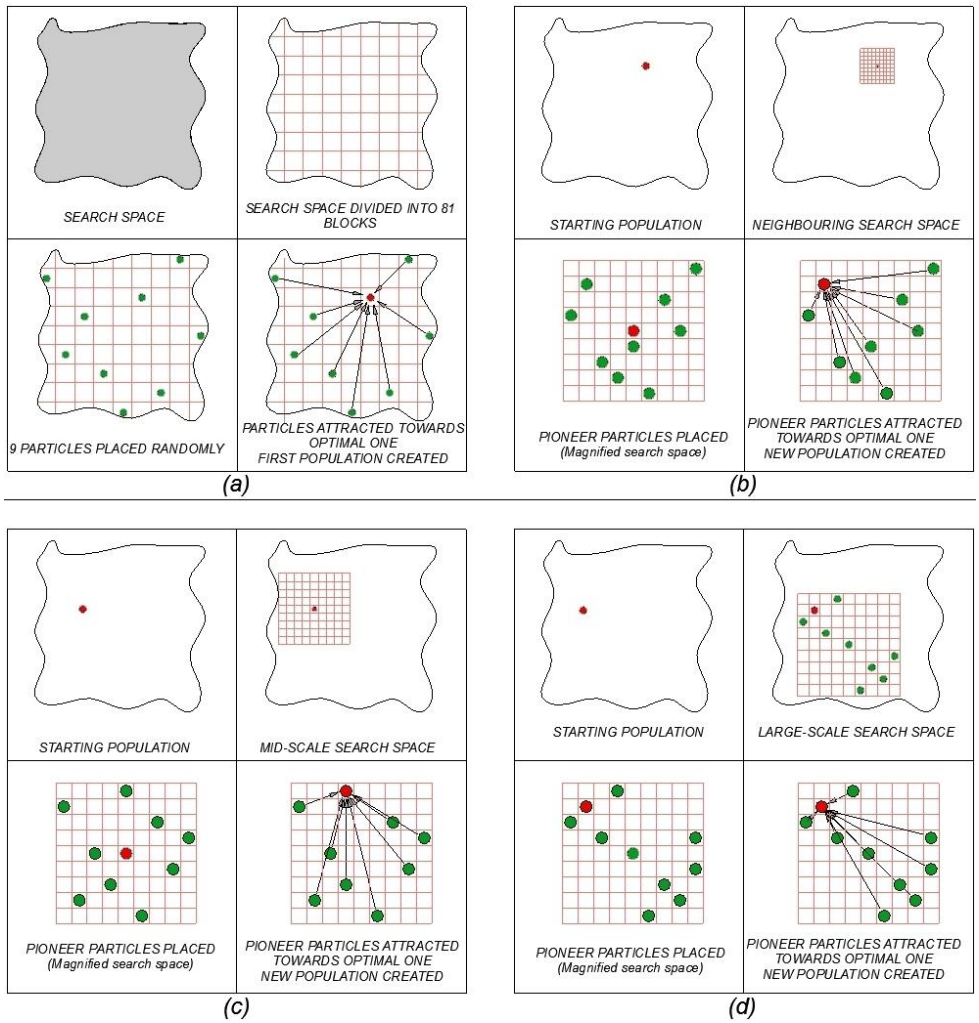
$$\begin{aligned} & x_j^{(g)} = RST(l^{(g)}, u^{(g)}, D, N_{pop}) \\ & \text{where} \\ & l_i^{(g)}, u_i^{(g)} \in \left[x_{birth,i}^{(g)} \cdot (1 - f_{pat}), x_{birth,i}^{(g)} \cdot (1 + f_{pat}) \right] \end{aligned} \tag{1.7}$$

όπου η παράμετρος f_{pat} ορίζει το εύρος του υπερκύβου κάθε μετακίνησης. Η διατύπωση του τρόπου επιλογής τρόπου μετακίνησης για κάθε γενιά είναι:

$$x_{j,k}^{(g)} = \begin{cases} RST\left(\left[x_{birth,i}^{(g)} \cdot (1-f_{nb}), x_{birth,i}^{(g)} \cdot (1+f_{nb})\right], D, N_{pop}\right), if\ k=1 \\ else \begin{cases} RST(L,U,D,N_{pop}), ifFE > FE_{un} \\ else \begin{cases} RST\left(\left[x_{birth,i}^{(g)} \cdot (1-f_{ms}), x_{birth,i}^{(g)} \cdot (1+f_{ms})\right], D, N_{pop}\right), \exists j,\ f(x_{j,k-1}^{(g)}) < f(x_{birth}^{(g)}) \\ else \begin{cases} RST\left(\left[x_{birth,i}^{(g)} \cdot (1-f_{ls}), x_{birth,i}^{(g)} \cdot (1+f_{ls})\right], D, N_{pop}\right), \text{if } r < pr \\ MEM, otherwise \end{cases} \end{cases} \end{cases} \end{cases} \quad (1.8)$$

$$i = 1, 2, \ldots, D,\ j = 1, 2, \ldots, N_{pop}$$

Μία σχηματική απεικόνιση του τρόπου μετακίνησης σε δισδιάστατο πρόβλημα με εννιά άτομα φαίνεται στην ακόλουθη εικόνα:



Η μετακίνηση με εκμετάλλευση της μνήμης πραγματοποιείται επιλέγοντας μία αποθηκευμένη λύση και πραγματοποιώντας ακτινικές μετακινήσεις ανά μεταβλητή.

Για την εκτίμηση της απόδοσης του PBA, πραγματοποιήθηκαν σειρά από τεστ που προτείνονται στη διεθνή βιβλιογραφία αλλά και ανάλυση ευαισθησίας ως προς τις παραμέτρους. Στα περισσότερα τεστ παρουσιάζεται και σύγκριση του PBA με

διάφορους σύγχρονους μεταευρετικούς αλγόριθμους. Ενδεικτικά, κάποιες από τις συναρτήσεις που χρησιμοποιήθηκαν στα τεστ είναι:

Test Function		Domain	Optimum	Name
$f_1(x)$	$f(x)=\sum_{i=1}^{n} x_i^2$	[-100, 100]^D	0	*Sphere*
$f_2(x)$	$f(x)=\sum_{i=1}^{n}[-x_i \sin(\sqrt{\lvert x_i \rvert})]$	[-500, 500]^D	-418.9829D	*Schwefel*
$f_3(x)$	$f(x)=\frac{1}{4000}\sum_{i=1}^{n} x_i^2-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	[-600, 600]^D	0	*Griewank*
$f_4(x)$	$f(x)=10n+\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)]$	[-100, 100]^D	0	*Rastrigin*
$f_5(x)$	$f(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(1-x_i)^2]$	[-10, 10]^D	0	*Rosenbrock*
$f_6(x)$	$f(x)=-20\exp(-0.20\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))+20+\exp(1)$	[-32.768, 32.768]^D	0	*Ackley*
$f_7(x)$	$f(x)=\sum_{i=1}^{n}\left(\sum_{k=0}^{k_{\max}}\left[a^k\cos(2\pi b^k(x_i+0.5))\right]\right)-n\sum_{k=0}^{k_{\max}}[a^k\cos(2\pi b^k*0.5)]$ $a=0.5,\ b=3,\ k_{\max}=20$	[-0.5, 0.5]^D	0	*Weierstrass*
$f_8(x)$	$f(x)=\sum_{i=1}^{n}(\sum_{j=1}^{i}x_j)^2$	[-100, 100]^D	0	*Quadric*
$f_9(x)$	$f(x)=\sum_{i=1}^{n}(\mathrm{x}_i+0.5)^2$	[-100, 100]^D	0	*Step*
$f_{10}(x)$	$f(x)=-20\exp(-0.20\sqrt{\frac{1}{n}\sum_{i=1}^{n}y_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi y_i))+20+\exp(1),$ $y=M\cdot x$	[-32.768, 32.768]^D	0	*Rotated Ackley*
$f_{11}(x)$	$f(x)=\frac{1}{4000}\sum_{i=1}^{n}y_i^2-\prod_{i=1}^{n}\cos(\frac{y_i}{\sqrt{i}})+1,$ $y=M\cdot x$	[-600, 600]^D	0	*Rotated Griewank*
$f_{12}(x)$	$f(x)=10n+\sum_{i=1}^{n}[\mathrm{y}_i^2-10\cos(2\pi y_i)],$ $y=M\cdot x$	[-5.12, 5.12]^D	0	*Rotated Rastrigin*
$f_{13}(x)$	$f(x)=10n+\sum_{i=1}^{n}[\mathrm{z}_i^2-10\cos(2\pi z_i)]-330,$ $z=(x-o)\cdot M$	[-5.12, 5.12]^D	-330	*Shifted – Rotated Rastrigin*

Κάποια ενδεικτικά αποτελέσματα παρουσιάζονται στον πίνακα που ακολουθεί:

Set	Sphere	Schwefel's	Griewank's	Rastrigin's	Rosenbrock's	Ackley's
1	1.19E-33	-12478.46	**0.00E+00**	**0.00E+00**	2.59E+01	7.99E-15

2	1.01E-83	-12400.32	1.70E-02	**0.00E+00**	2.88E+01	4.44E-15
3	4.98E-145	-12559.84	**0.00E+00**	**0.00E+00**	2.88E+01	**8.88E-16**
4	2.88E-25	-12460.17	4.02E-02	**0.00E+00**	2.86E+01	7.19E-14
5	4.36E-18	-12301.16	**0.00E+00**	**0.00E+00**	2.60E+01	6.56E-10
6	2.48E-77	-12414.98	1.93E-02	**0.00E+00**	2.59E+01	4.44E-15
7	3.84E-13	-12416.15	7.76E-13	1.51E-13	2.88E+01	1.25E-07
8	1.47E-25	-12533.16	**0.00E+00**	**0.00E+00**	2.71E+01	7.19E-14
9	1.59E-98	-12453.65	**0.00E+00**	**0.00E+00**	2.86E+01	2.58E-14
10	1.08E-07	-12558.81	6.70E-02	4.52E-08	2.80E+01	4.87E-05
11	6.01E-65	-12514.33	1.26E-02	**0.00E+00**	2.58E+01	7.99E-15
12	8.60E-45	-12294.99	1.09E-02	**0.00E+00**	**2.55E+01**	7.99E-15
13	5.11E-24	-12332.07	**0.00E+00**	**0.00E+00**	2.70E+01	3.42E-13
14	5.30E-21	-12425.66	**0.00E+00**	**0.00E+00**	2.88E+01	1.43E-11
15	4.90E-26	-12331.18	1.33E-02	**0.00E+00**	2.89E+01	5.77E-14
16	4.85E-36	-12437.18	**0.00E+00**	**0.00E+00**	2.79E+01	4.44E-15
17	9.55E-23	-12355.17	**0.00E+00**	**0.00E+00**	2.61E+01	1.24E-12
18	9.05E-07	-12540.62	3.40E-06	5.08E-07	2.69E+01	2.00E-04
19	9.31E-49	**-12561.68**	2.85E-02	**0.00E+00**	2.86E+01	4.44E-15
20	1.04E-15	-12353.39	1.01E-02	4.97E-14	2.56E+01	3.43E-07
21	**3.47E-206**	-12529.82	**0.00E+00**	**0.00E+00**	2.86E+01	3.44E+00
22	3.89E-18	-12353.25	**0.00E+00**	**0.00E+00**	2.80E+01	3.50E-10
23	5.69E-09	-12555.03	1.61E-07	1.60E-08	2.71E+01	2.30E-05
24	8.14E-16	-12530.70	1.33E-15	**0.00E+00**	2.70E+01	5.63E-09
25	1.89E-75	-12474.81	**0.00E+00**	**0.00E+00**	2.88E+01	3.58E+00
Average	4.07E-08	-12446.66	8.75E-03	2.27E-08	2.75E+01	2.81E-01
St. Deviation	1.78E-07	86.79	1.58E-02	9.94E-08	1.22E+00	9.53E-01
Minimum	3.47E-206	-12561.68	0.00E+00	0.00E+00	2.55E+01	8.88E-16
Maximum	9.05E-07	-12294.99	6.70E-02	5.08E-07	2.89E+01	3.58E+00

Επίσης, εφαρμόστηκε και το τεστ του διαγωνισμού για single objective unconstrained optimization του CEC 2014 [127] και τα αποτελέσματα παρουσιάζονται λεπτομερώς στο αντίστοιχο κεφάλαιο της διδακτορικής διατριβής.

3. Μέθοδοι βαθιάς μηχανικής μάθησης

Η μηχανική μάθηση μπορεί να περιγραφεί ως το σύνολο των διαδικασιών εκείνων που δίνουν στον υπολογιστή την ικανότητα να παίρνει αποφάσεις και να δρα βασιζόμενος σε κανόνες που ορίζονται από ΄σωστές΄ αποφάσεις ή δράσεις αλλά και να προσαρμόζεται με βάση τις συνέπειες των σωστών ή λανθασμένων αποφάσεων σε μια επαναληπτική διαδικασία [137]. Η μηχανική μάθηση χρησιμοποιείται πλέον σε πληθώρα εφαρμογών όπως διάγνωση ασθενειών, υπολογιστική όραση, αναγνώριση δομικών προβλημάτων, κλπ. [110, 41,23].

Με βάση τη διαδικασία εκπαίδευσης, οι τεχνικές μηχανικής μάθησης μπορούν να κατηγοριοποιηθούν ως εξής [137]:

- Επιβλεπόμενη μάθηση (Supervised learning)
- Μη-επιβλεπόμενη μάθηση (Unsupervised learning)
- Ημι-επιβλεπόμενη μάθηση (Semi-supervised learning)
- Ενισχυόμενη μάθηση (Reinforcement learning)
- Εξελικτική μάθηση (Evolutionary learning)

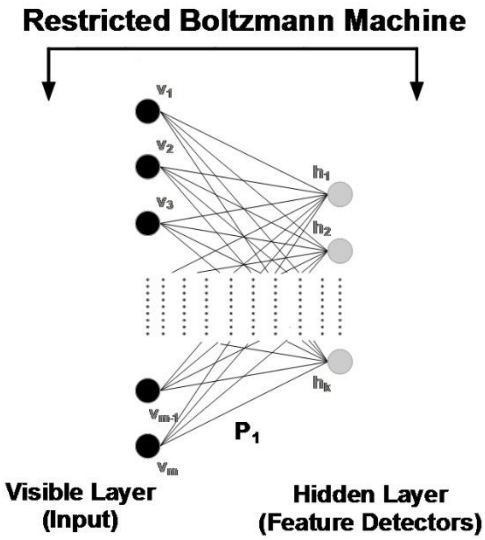
Στα πλαίσια της παρούσας διδακτορικής διατριβής γίνεται χρήση τεχνικών βαθιών νευρωνικών δικτύων και επιβλεπόμενης μάθησης. Πιο συγκεκριμένα, γίνεται χρήση των Deep Belief Networks και Deep Convolutional Neural Networks.

3.1 Deep Belief Networks

Τα Deep Belief Networks (DBN) είναι βαθιά νευρωνικά δίκτυα που μορφώνονται από την σειριακή σύνδεση πολλαπλών δικτύων Restricted Boltzmann Machines (RBM) [73].

Τα RBM [196] είναι πιθανοτικά γραφικά μοντέλα, που αναφέρονται και ως στοχαστικά νευρωνικά δίκτυα. Χρησιμοποιούνται κυρίως για την ανακάλυψη ιδιοτήτων μιας άγνωστης κατανομής πιθανότητας με χρήση δειγμάτων αυτής και μοντελοποίησής της σε διαφορετικό χώρο.

Η αρχιτεκτονική των δικτύων αυτών διαμορφώνεται από δύο layer, το ορατό (visible) που αντιστοιχεί στα δεδομένα εισόδου και το κρυφό (hidden) που αντιστοιχεί στους ανιχνευτές χαρακτηριστικών. Οι κόμβοι του ορατού layer είναι συνδεδεμένοι συμμετρικά με όλους τους κόμβους του κρυφού layer ενώ δεν υπάρχουν συνδέσεις μεταξύ κόμβων που ανήκουν στο ίδιο layer. Μία ενδεικτική απεικόνιση ενός RBM παρουσιάζεται στην παρακάτω εικόνα:



Σε κάθε κατάσταση του δικτύου αντιστοιχεί και μια ενέργεια, η οποία μπορεί να υπολογιστεί ως εξής [39]:

$$E(v,h) = -\sum_{k=1}^{\mathrm{k\,max}} a_k v_k - \sum_{l=1}^{\mathrm{l\,max}} b_l h_l - \sum_{k=1}^{\mathrm{k\,max}} \sum_{l=1}^{\mathrm{l\,max}} v_k h_l w_{k,l} \qquad (1.9)$$

Η ενέργεια αυτή αποτελεί δείκτη της ποιότητας της κατάστασης του δικτύου. Αντίστοιχα, η συνδυασμένη πιθανότητα του δικτύου για κάθε πιθανό ζεύγος εισόδων και εξόδων του δικτύου υπολογίζεται από τον νόμο του Boltzmann ως εξής [131]:

$$p(\mathrm{v},\mathrm{h}) = \frac{1}{Z} e^{-E(\mathrm{v},\mathrm{h})} \qquad \text{where } Z = \sum_{v,h} e^{-E(\mathrm{v},\mathrm{h})} \text{(intractable)}$$

$$p(v,h) = \frac{1}{Z} e^{\sum_k a*v} e^{\sum_l b*h} e^{\sum_k \sum_l v*h*w} = \frac{1}{Z} \prod_k e^{a*v} \prod_l e^{\mathrm{b*h}} \prod_k \prod_l e^{v*h*w} \qquad (1.10)$$

Όμοια, μπορούν να υπολογιστούν και οι κάτωθι δεσμευμένες πιθανότητες ως εξής:

$$p(v \mid h) = \prod_{k=1}^{\mathrm{k\,max}} p(v_k \mid h) \quad \text{and} \quad p(\mathrm{h} \mid \mathrm{v}) = \prod_{l=1}^{\mathrm{l\,max}} p(\mathrm{h}_l \mid \mathrm{v}) \qquad (1.11)$$

Αντίστοιχα, η πιθανότητα να είναι ενεργός κάποιος κόμβος είτε του ορατού είτε του κρυφού layer δίνεται από:

$$p(h_l = 1|v) = \frac{e^{-(\mathrm{b}_l + \sum_v v_k w_{k.l})}}{1 + e^{-(\mathrm{b}_l + \sum_v v_k w_{k.l})}} = \frac{e^{\mathrm{b}_l + \sum_v v_k w_{k.l}}}{1 + e^{\mathrm{b}_l + \sum_v v_k w_{k.l}}} * \frac{e^{-(\mathrm{b}_l + \sum_v v_k w_{k.l})}}{e^{-(\mathrm{b}_l + \sum_v v_k w_{k.l})}} = \frac{1}{1 + e^{-(b_l + \sum_v v_k w_{k.l})}} \qquad \text{(logistic function)}$$

(1.12)

$$p(\mathrm{v}_k = 1|\mathrm{h}) = \frac{1}{1 + e^{-(\alpha_k + \sum_h h_l w_{k.l})}} \qquad \text{(logistic function)} \qquad (1.13)$$

Η εκπαίδευση ενός RBM γίνεται με στόχο τη μεγιστοποίηση του log-likelihood [74]:

$$\sum_{k=1}^{k_{\max}} \log P(\mathrm{v}^{(k)}) \qquad (1.14)$$

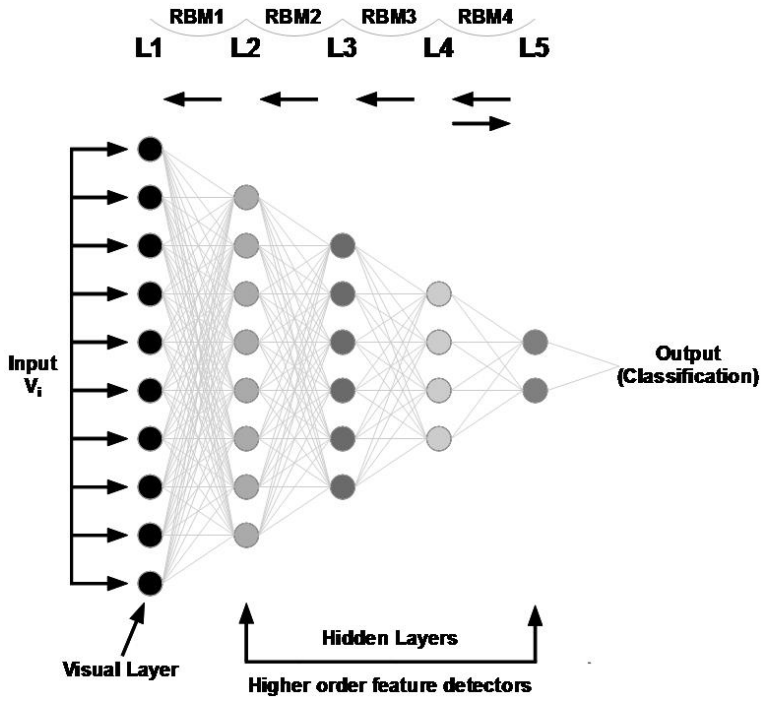
με χρήση κάποιου αλγόριθμου βελτιστοποίησης, συνηθέστερα του Gradient ascent [50] και τη μεθοδολογία Contrastive Divergence [70,71]. Επιπλέον, για την εκπαίδευση του RBM, μπορεί να εκτιμηθεί και η παράγωγος του λογαρίθμου της πιθανότητας ενός διανύσματος εισόδου ως προς το βάρος $w_{k,l}$ ως εξής [77]:

$$\frac{\partial \log p(v)}{\partial w_{k,l}} = \langle v_k h_l \rangle_{\text{input}} - \langle v_k h_l \rangle_{\text{model}} \qquad \text{(expected average values)} \qquad (1.15)$$

όπου $\langle v_k h_l \rangle_{\text{input}}$ και $\langle v_k h_l \rangle_{\text{model}}$ εκφράζουν τη συχνότητα με την οποία τα v_k, h_l είναι ενεργά ταυτόχρονα στο μοντέλο, όπως προκύπτει από τα δεδομένα εισόδου και στο ανακατασκευασμένο μοντέλο.
Ακολούθως, γίνεται ενημέρωση των βαρών μέσω της σχέσης:

$$w_{kl}^{new} = w_{kl} + e \cdot \Delta w_{kl} \qquad e\text{: learning rate} \qquad (1.16)$$

Η εκπαίδευση των DBN βασίζεται σε μία διβηματική μέθοδο που προτάθηκε την τελευταία δεκαετία, καθώς παλιότερα δεν υπήρχε εκπαίδευση με ικανοποιητικά αποτελέσματα [75]. Στο πρώτο στάδιο της μεθόδου, κάθε RBM που υπάρχει στο DBN εκπαιδεύεται ανεξάρτητα από τα υπόλοιπα με μη-επιβλεπόμενη εκπαίδευση. Όταν ολοκληρωθεί αυτό το βήμα, μορφώνεται το DBN από το προ-εκπαιδευμένα RBM και το όλο δίκτυο πλέον, εκπαιδεύεται με την χρήση του Backpropagation [176] και Conjugate Gradient [68] με εκπαιδευόμενη μάθηση [74, 75].
Ένα γενικό μοντέλο ενός δικτύου DBN αναπαρίσταται στην παρακάτω εικόνα:

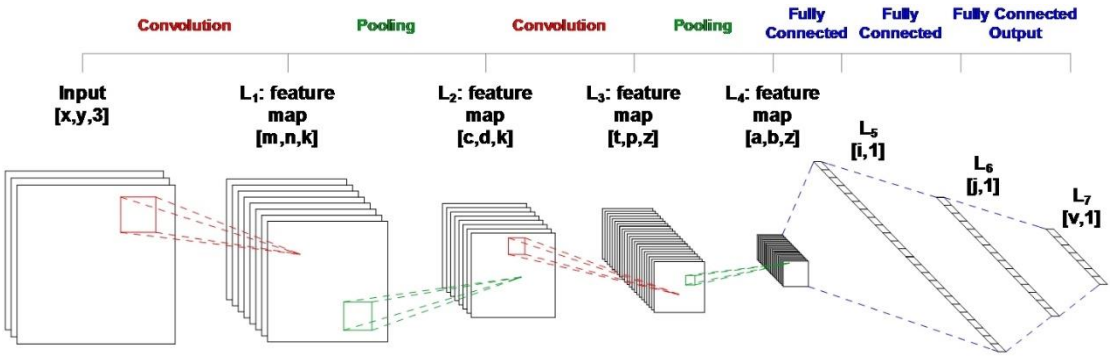


3.2 Convolutional Neural Networks

Τα Convolutional Neural Networks (CNN), εμπνευσμένα από τον τρόπο λειτουργίας των οργάνων όρασης των ζώων [82,83], αποτελούν ίσως τα πιο ευρέως χρησιμοποιούμενα βαθιά νευρωνικά δίκτυα. Αν και σε πρώτη μορφή παρουσιάστηκαν αρκετά χρόνια πριν [56,118,119], η απόδοσή τους σε προβλήματα αναγνώρισης αντικειμένων σε εικόνες τα τελευταία χρόνια, τα έφερε στο προσκήνιο [111]. Έκτοτε έχουν προταθεί διάφορες αρχιτεκτονικές (LeNet, ZFNet, GoogLeNet) με αξιοσημείωτα αποτελέσματα [121,226,203].
Η αρχιτεκτονική των CNN αποτελείται από εν σειρά σύνδεση διαφόρων τύπων δικτύων. Οι βασικοί τύποι που χρησιμοποιούνται είναι:

- Convolutional layers,
- Pooling layers και
- Fully connected layers

Μία τυπική μορφή ενός CNN, όπως προτάθηκε από τον LeCun (δίκτυο LeNet), φαίνεται στην παρακάτω εικόνα:



Τα convolutional layers χρησιμοποιούνται ως ανιχνευτές χαρακτηριστικών στα δεδομένα εισόδου. Η παραπάνω διαδικασία εκτελείται με τη χρήση φίλτρων που διατρέχουν τα δεδομένα εισόδου πραγματοποιώντας την συνέλιξη. Τα pooling layers συνήθως ακολουθούν τα convolutional και χρησιμοποιούνται για την μείωση του μεγέθους των δεδομένων εισόδου. Υπάρχουν τρεις τύποι pooling [63]:

- Max pooling,
- Average pooling και
- L^2-norm pooling.

Ο ευρύτερα χρησιμοποιούμενος τύπος pooling είναι ο max pooling, αν και η επιλογή εξαρτάται από το πρόβλημα αυτό καθαυτό αλλά και από τον τύπο των δεδομένων εισόδου. Τα fully connected layers χρησιμοποιούνται αντίστοιχα με το πρόβλημα που καλείται το CNN να διαχειριστεί (classification, regression, κλπ.) και είναι όμοια με τυπικά layers κλασσικών νευρωνικών δικτύων.

Τα CNN εκπαιδεύονται συνήθως με την χρήση του διαδεδομένου αλγόριθμου Backpropagation καθώς με την χρήση των convolution και pooling layers μειώνεται σημαντικά το μέγεθος των fully connected layers και γίνονται διαχειρίσιμα.

4. Εφαρμοσμένη βαθιά μηχανική μάθηση στη βελτιστοποίηση τοπολογίας

Το συγκεκριμένο κεφάλαιο της διδακτορικής διατριβής είναι αφιερωμένο στην αναζήτηση νέων μεθόδων για την επιτάχυνση της βελτιστοποίησης τοπολογίας, η οποία είναι ιδιαίτερα απαιτητική από την άποψη υπολογιστικού φόρτου και χρόνου, ακόμα και σε επίπεδο ερευνητικής και όχι πραγματικού μεγέθους δουλειάς. Παρά το γεγονός πως οι διαθέσιμοι υπολογιστικοί πόροι αυξάνονται, η αύξηση της πολυπλοκότητας και του μεγέθους των προβλημάτων είναι ταχύτερη οδηγώντας σε αύξηση του ενδιαφέροντος για επιτάχυνση των διαδικασιών αυτών. Κάποιοι προτεινόμενοι τρόποι αντιμετώπισης του προβλήματος επικεντρώνονται στη χρήση παράλληλου προγραμματισμού σε CPU ή και GPU [16,24,45,138] ενώ άλλοι στα μοντέλα μειωμένης τάξης, κλπ.

Η βελτιστοποίηση τοπολογίας είναι μια μαθηματική μέθοδος που αποσκοπεί στην εύρεση της βέλτιστης τοποθέτησης συγκεκριμένου ποσοστού υλικού εντός συγκεκριμένου χωρίου υπό συγκεκριμένες συνθήκες στήριξης και φόρτισης με κριτήριο την στατική απόδοση του συστήματος [194]. Το συνηθέστερα χρησιμοποιούμενο κριτήριο στατικής απόδοσης του συστήματος είναι το Compliance. Μία από τις ευρέως χρησιμοποιούμενες μεθόδους στην Βελτιστοποίηση Τοπολογίας είναι η Solid Isotropic Material with Penalization (SIMP), η οποία συνδέει την πυκνότητα με το μέτρο ελαστικότητας και παρουσιάστηκε τη δεκαετία του 1990 [12,229,147]. Η διατύπωση ενός προβλήματος βελτιστοποίησης τοπολογίας είναι η εξής:

$$
\begin{aligned}
&Minimize \quad C(x) = F^T * U(x) \\
&with \quad respect \quad to: \\
&\qquad K(x) * U(x) = F \\
&\qquad \frac{V(x)}{V_0} = V_t \\
&\qquad 0 < x \leq 1
\end{aligned}
\tag{1.17}
$$

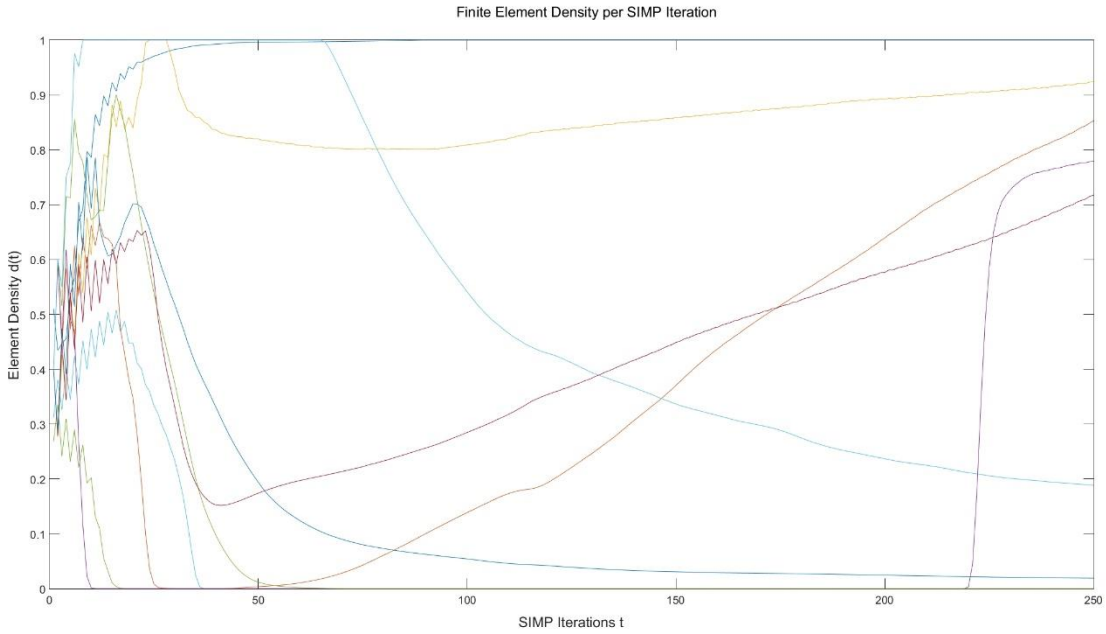
όπου C(x) είναι το compliance για συγκεκριμένο διάνυσμα πυκνοτήτων, K(x) το μητρώο δυσκαμψίας, F και U(x) τα μητρώα δυνάμεων και μετατοπίσεων, V(x),V_0,V_t είναι ο όγκος με βάση το διάνυσμα πυκνοτήτων x, ο αρχικός όγκος και ο επιθυμητός όγκος, αντίστοιχα. Το μέτρο ελαστικότητας συνδέεται με την πυκνότητα κάθε πεπερασμένου στοιχείου μέσω της παρακάτω σχέσης:

$$E_x(x_i) = x_i^p E^0 \Leftrightarrow K_x(x_i) = x_i^p K^0 \tag{1.18}$$

Στην παρούσα διδακτορική διατριβή προτείνεται μία νέα μέθοδος επιτάχυνσης των διαδικασιών επίλυσης προβλημάτων βελτιστοποίησης τοπολογίας, μειώνοντας δραματικά τον απαιτούμενο υπολογιστικό φόρτο τους. Η νέα μέθοδος που

ονομάζεται DL-TOP βασίζεται στον συνδυασμό σύγχρονων μεθόδων βαθιών νευρωνικών δικτύων και συγκεκριμένα των DBN και της SIMP. Η κεντρική ιδέα πάνω στην οποία δημιουργήθηκε το DL-TOP απαρτίζεται από τα εξής:

- Η βελτιστοποίηση τοπολογίας είναι ιδιαίτερα απαιτητική σε υπολογιστικό φόρτο και η απαίτηση αυτή αυξάνεται σε συνάρτηση με το πλήθος των πεπερασμένων στοιχείων του πλέγματος
- Τα DBN έχουν αξιοσημείωτες δυνατότητες αναγνώρισης προτύπων που βρίσκονται κρυμμένα σε μεγάλες βάσεις δεδομένων και δεν είναι ορατά από τον χρήστη χωρίς εφαρμογή της μεθόδου
- Κάθε πεπερασμένο στοιχείο παρουσιάζει μεταβολή στην πυκνότητά του σε κάθε επανάληψη της SIMP. Ενδεικτική διακύμανση της πυκνότητας διάφορων πεπερασμένων στοιχείων μπορεί να παρατηρηθεί στην παρακάτω εικόνα όπου ο άξονας των x αντιστοιχεί στις επαναλήψεις της SIMP, ο y στην τιμή της πυκνότητας και με διαφορετικό χρώμα σημειώνονται τα διάφορα πεπερασμένα στοιχεία:



- Η προτεινόμενη μεθοδολογία πρέπει να είναι ανεξάρτητη των παραμέτρων του προβλήματος (γεωμετρία, συνθήκες στήριξης και φόρτισης, τελικός όγκος, κλπ.). Με τον τρόπο αυτό θα αποφευχθεί η αναγκαιότητα επανεκπαίδευσης του νευρωνικού δικτύου σε κάθε εφαρμογή. Κάτι τέτοιο θα ήταν ασύμφορο χρονικά και υπολογιστικά.

Η προτεινόμενη μεθοδολογία DL-TOP αποτελείται από τα παρακάτω στάδια:

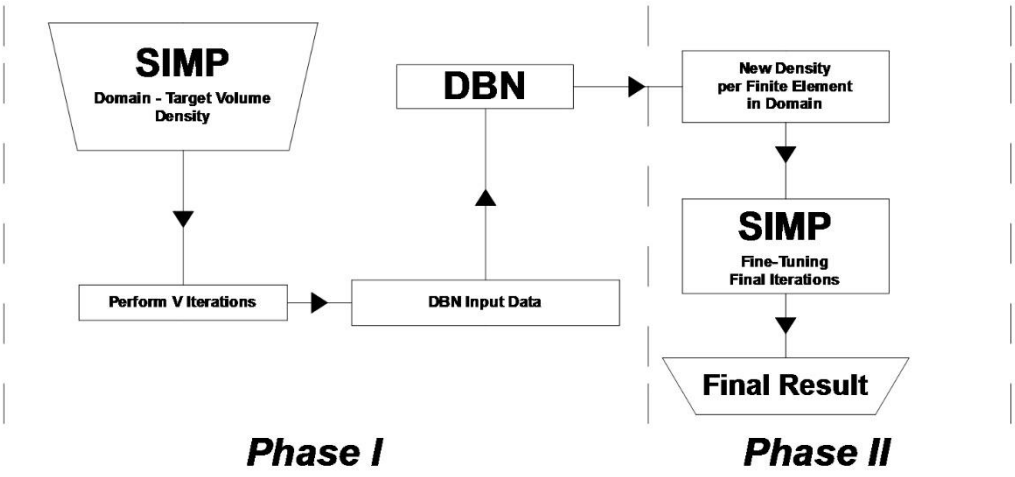
Φάση Ι

- Ορισμός του προβλήματος βελτιστοποίησης τοπολογίας
- Εκτέλεση περιορισμένου αριθμού επαναλήψεων της SIMP
- Μόρφωση διανύσματος εισόδου του DBN ανά πεπερασμένο στοιχείο

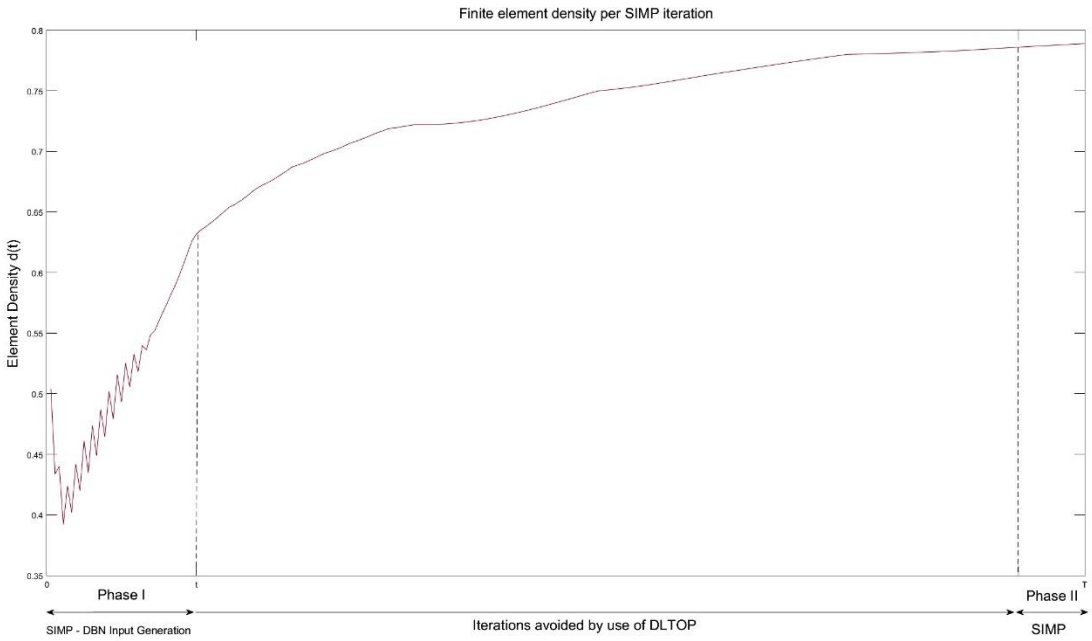
Φάση ΙΙ

- Εκτίμηση πιθανής τελικής πυκνότητας κάθε πεπερασμένου στοιχείου από το DBN
- Μόρφωση του χωρίου με βάση τις προτεινόμενες τιμές πυκνότητας ανά πεπερασμένο στοιχείο
- Χρήση της SIMP για να πραγματοποιήσει fine-tuning στο αποτέλεσμα του DBN

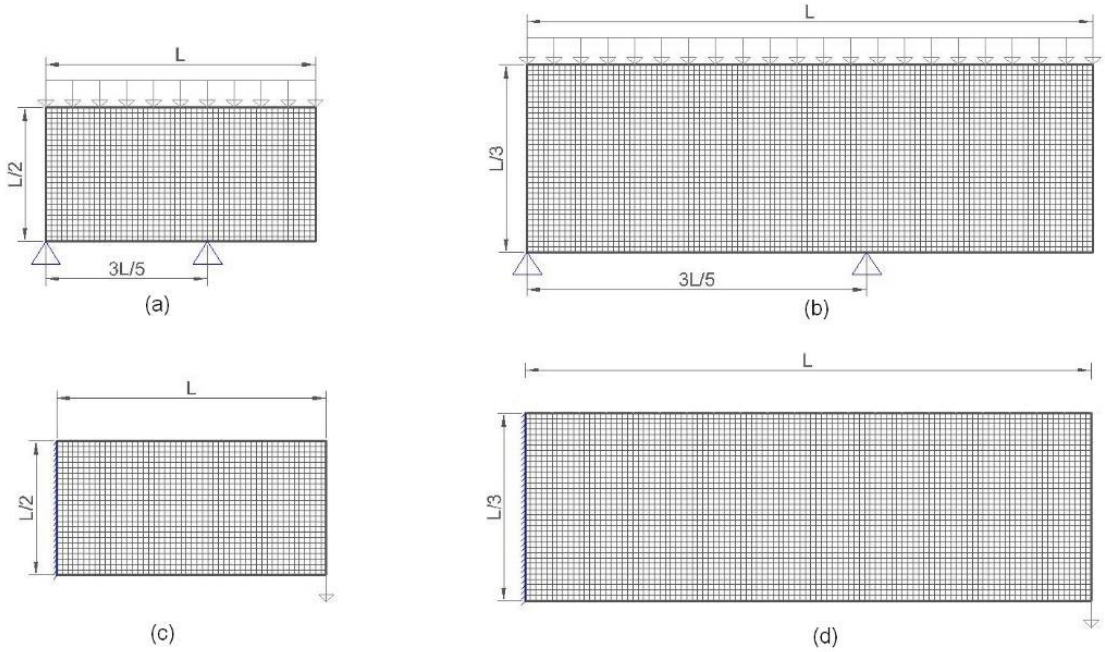
Το διάγραμμα ροής του DL-TOP παρουσιάζεται στην ακόλουθη εικόνα:



Συγκεκριμένα, το DBN εκπαιδεύεται και χρησιμοποιείται για να πραγματοποιεί ένα διακριτό άλμα από την πυκνότητα κάθε πεπερασμένου στοιχείου στις πρώτες επαναλήψεις σε μια τιμή κοντά στην τελική που θα υπολόγιζε η SIMP από μόνη της, μειώνοντας δραστικά το πλήθος των απαιτούμενων επαναλήψεων της SIMP. Το παραπάνω φαίνεται και στην ακόλουθη εικόνα:

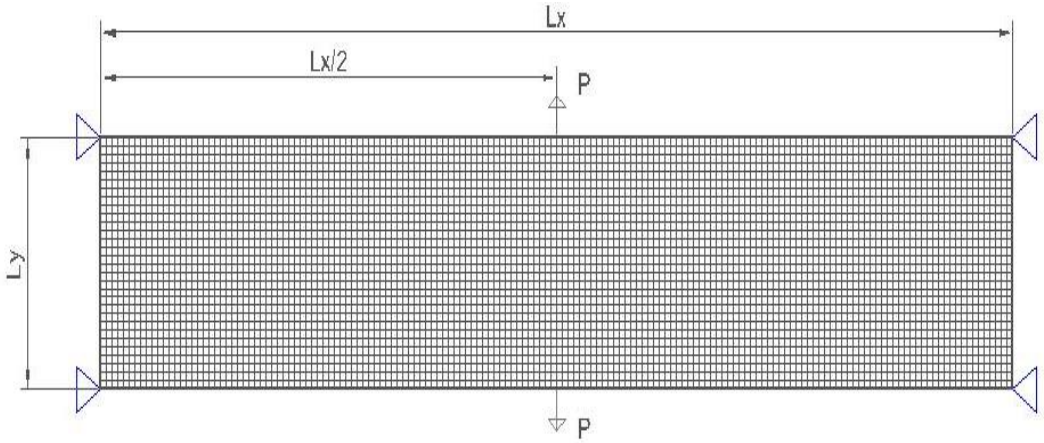


Για να μπορέσει το DBN να προσφέρει αυτήν την πληροφορία, είναι απαραίτητη η εκπαίδευσή του σε παρόμοιους συσχετισμούς, δηλαδή δεδομένα εισόδου που είναι χρονο-ιστορίες τιμών πυκνοτήτων πεπερασμένων στοιχείων ανά επανάληψη της SIMP (για μικρό πλήθος επαναλήψεων) και τελικές τιμές πυκνότητας των πεπερασμένων αυτών στοιχείων. Για τον λόγο αυτό κατασκευάστηκαν δύο βάσεις δεδομένων από δύο κλασικά παραδείγματα βελτιστοποίησης τοπολογίας, μια μονοπροέχουσα αμφιέρειστη δοκό και έναν πρόβολο όπως φαίνονται στην παρακάτω εικόνα:

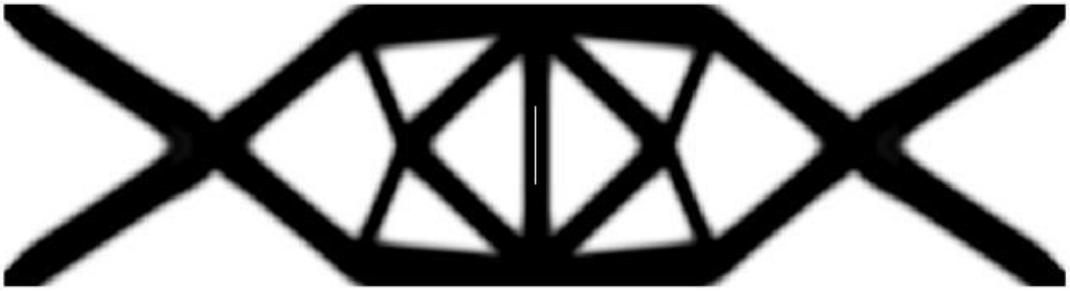


με δύο αναλογίες ύψους/μήκους και οκτώ διαφορετικά πλήθη διακριτοποιήσεων κυμαινόμενα από 1000 έως 100.000 πεπερασμένα στοιχεία. Συνολικά, κάθε βάση περιέχει περίπου 400.000 διαφορετικούς συσχετισμούς πυκνοτήτων. Συνεπώς, το DBN εκπαιδεύτηκε σε δύο διαφορετικές βάσεις ξεχωριστά. Ως κατηγορίες κλάσεων στην έξοδο του DBN, εξετάζονται δύο διαφορετικές περιπτώσεις, τρεις και έντεκα διαφορετικές κλάσεις. Για να εκτιμηθεί η συμπεριφορά της μεθοδολογίας DL-TOP, δοκιμάστηκε σε σειρά παραδειγμάτων δύο διαστάσεων, τριών διαστάσεων, με δομημένο και μη δομημένο πλέγμα. Η εκτίμηση της απόδοσης έγινε ως προς το τελικό αποτέλεσμα μείωσης επαναλήψεων της SIMP, γενικότητα εφαρμογής σε διαφορετικά προβλήματα και σταθερότητα της μεθοδολογίας ως προς τις τιμές των παραμέτρων της.

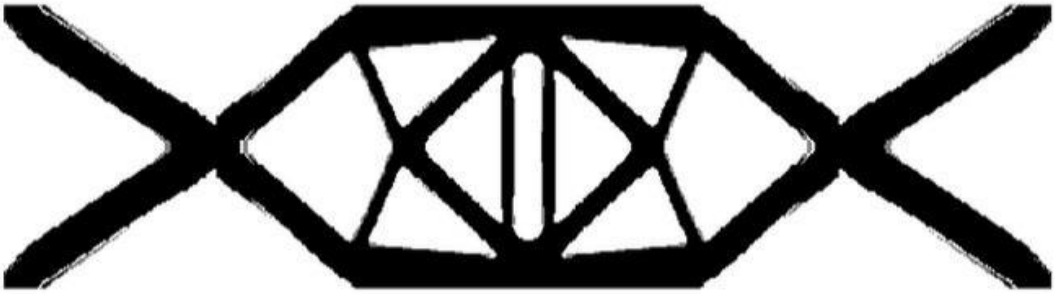
Το πρώτο παράδειγμα είναι μία δοκός με τις στηρίξεις και τις φορτίσεις που φαίνεται στην παρακάτω εικόνα, δομημένο πλέγμα, πλήθος πεπερασμένων στοιχείων 120.000 και τελικό στόχο όγκου ίσο με 40%.



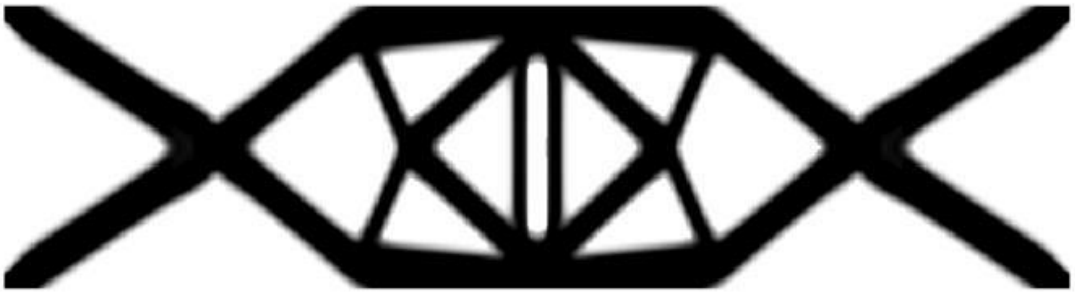
Η λύση που προτείνει η SIMP χωρίς το DBN προκύπτει μετά από 372 επαναλήψεις και τιμή αντικειμενικής συνάρτησης ίση με 85,99. Η τελική τοπολογία που προτείνει είναι η εξής:



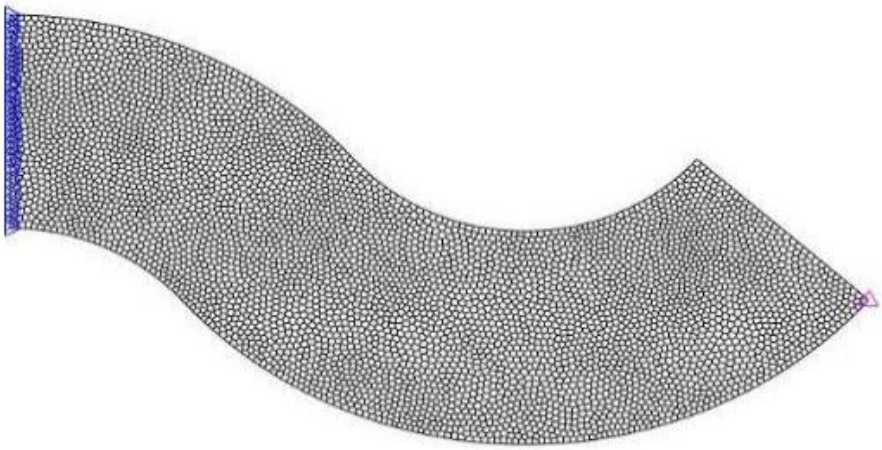
Η προτεινόμενη τοπολογία στην έξοδο του DBN έχοντας διαβάσει τις πρώτες 36 επαναλήψεις της SIMP έχει τιμή αντικειμενικής συνάρτησης ίση με 86,43 και εικόνα:



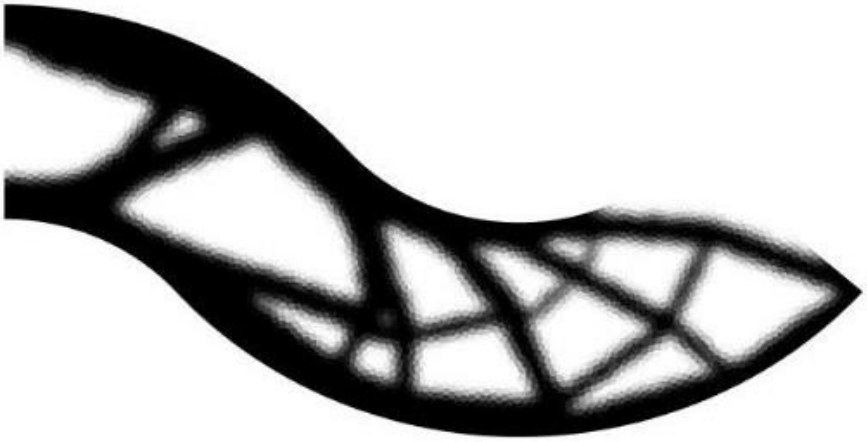
Το τελικό αποτέλεσμα της μεθοδολογίας DL-TOP μετά από το τελικό της στάδιο το οποίο είναι το fine-tuning της εξόδου του DBN από τη SIMP (43 επαναλήψεις) έχει τιμή αντικειμενικής συνάρτησης ίση με 85,66, ελάχιστα μικρότερη από αυτή της SIMP και οι συνολικές επαναλήψεις που χρειάστηκαν είναι ίσες με 43+36 = 79 επαναλήψεις, οι οποίες αντιστοιχούν στο 21% μόλις αυτών που χρειάζεται η SIMP από μόνη της. Το τελικό αποτέλεσμα τοπολογίας που προέκυψε από την μεθοδολογία DL-TOP φαίνεται στο ακόλουθο σχήμα:



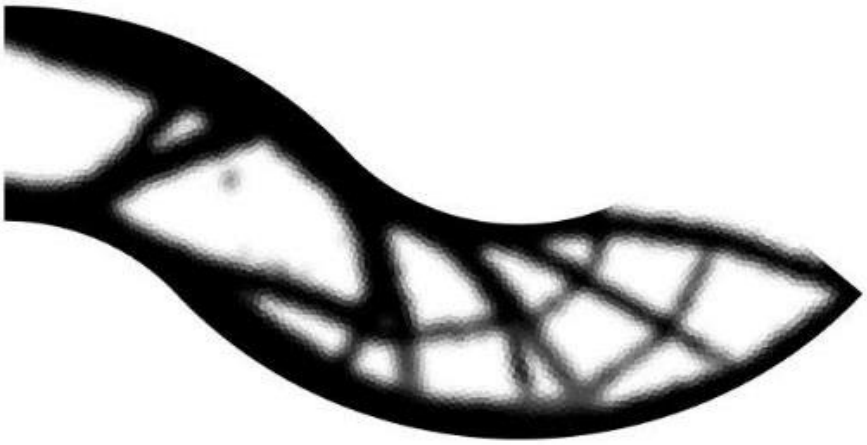
Το επόμενο παράδειγμα είναι ένας πρόβολος με την στήριξη και την φόρτιση που φαίνεται στην παρακάτω εικόνα, μη δομημένο πλέγμα, πλήθος πεπερασμένων στοιχείων 5.000 και τελικό στόχο όγκου ίσο με 40%.



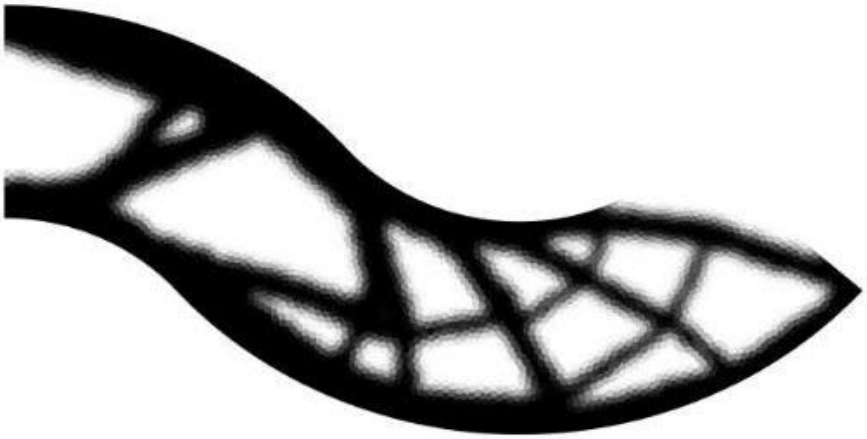
Η λύση που προτείνει η SIMP χωρίς το DBN προκύπτει μετά από 267 επαναλήψεις και τιμή αντικειμενικής συνάρτησης ίση με 391,19. Η τελική τοπολογία που προτείνει είναι η εξής:



Η προτεινόμενη τοπολογία στην έξοδο του DBN έχοντας διαβάσει τις πρώτες 36 επαναλήψεις της SIMP έχει τιμή αντικειμενικής συνάρτησης ίση με 442,65 και εικόνα:



Το τελικό αποτέλεσμα της μεθοδολογίας DL-TOP μετά από το τελικό της στάδιο το οποίο είναι το fine-tuning της εξόδου του DBN από τη SIMP (43 επαναλήψεις) έχει τιμή αντικειμενικής συνάρτησης ίση με 394,46, ελάχιστα μεγαλύτερη από αυτή της SIMP και οι συνολικές επαναλήψεις που χρειάστηκαν είναι ίσες με 30+36 = 66 επαναλήψεις, οι οποίες αντιστοιχούν στο 24,72% μόλις αυτών που χρειάζεται η SIMP από μόνη της. Το τελικό αποτέλεσμα τοπολογίας που προέκυψε από την μεθοδολογία DL-TOP φαίνεται στο ακόλουθο σχήμα:



Στο πλήρες κείμενο της διδακτορικής διατριβής παρουσιάζονται αρκετά περισσότερα παραδείγματα όπου παρατηρείται ακόμα καλύτερη συμπεριφορά της μεθοδολογίας, ενώ παρουσιάζεται σύγκριση των δύο διαφορετικών βάσεων εκπαίδευσης και μελέτη της σταθερότητας της μεθοδολογίας σε σχέση με τις παραμέτρους εκπαίδευσης των RBM.

5. Βαθιά νευρωνικά δίκτυα και μοντέλα μειωμένης τάξης

Το παρόν κεφάλαιο της διδακτορικής διατριβής επικεντρώνεται στην ανάπτυξη νέων μεθοδολογιών παραγωγής και χρήσης μοντέλων μειωμένης τάξης μέσω βαθιών νευρωνικών δικτύων στην βελτιστοποίηση κατασκευών. Πιο συγκεκριμένα, παρουσιάζονται τρεις μέθοδοι παραγωγής μοντέλων μειωμένης τάξης με συνδυασμό

βαθιών νευρωνικών δικτύων (DBN και CNN) και της μεθόδου βελτιστοποίησης τοπολογίας SIMP. Οι μεθοδολογίες αυτές ονομάζονται:

- DL-SCALE
- DLRM-TOP
- CN-TOP

Κάθε μία από αυτές προτείνει μια διαφορετική τεχνική παραγωγής και χρήσης μοντέλων μειωμένης τάξης που αποσκοπούν στη δραστική μείωση του χρόνου που απαιτεί η βελτιστοποίηση τοπολογίας με τις παραδοσιακές μεθόδους.

5.1 DL-SCALE – Upgrade μοντέλου μέσω deep learning

Στόχος του DL-SCALE είναι η επιτάχυνση της διαδικασίας βελτιστοποίησης ενός προβλήματος μέσω της εκμετάλλευσης γνώσης που παρέχεται από την εφαρμογή βελτιστοποίησης σε όμοια αλλά σημαντικά πιο αδρά διακριτοποιημένα μοντέλα. Το παραπάνω μπορεί να επιτευχθεί με την σειριακή εκτέλεση του DL-TOP σε συνδυασμό με ταυτόχρονη πύκνωση του πλέγματος του χωρίου προς εξέταση.

Ένα πρόβλημα βελτιστοποίησης τοπολογίας TOP_0 μπορεί να οριστεί ως εξής:

$$TOP_0 : [L_{xyz}^{(0)}, ne^{(0)}, S_c^{(0)}, P_c^{(0)}, V_t^{(0)}] \qquad (1.19)$$

όπου L_{XYZ}, ne, S_C, P_C, V_t είναι οι διαστάσεις του χωρίου, το πλήθος των πεπερασμένων στοιχείων, οι συνθήκες στήριξης και φόρτισης και ο επιθυμητός όγκος αντίστοιχα. Αν θεωρηθεί πως ο χρόνος εκτέλεσης βελτιστοποίησης τοπολογίας στο παράδειγμα αυτό είναι ίσος με t_0, είναι γνωστό πως ο χρόνος εκτέλεσης του ίδιου προβλήματος αλλά με διαφορετικό ne = $ne^{(1)}$ εξαρτάται από τον λόγο $\frac{ne^{(0)}}{ne^{(1)}}$. Για την ακρίβεια, η εξάρτηση αυτή παρουσιάζεται στον ακόλουθο πίνακα:

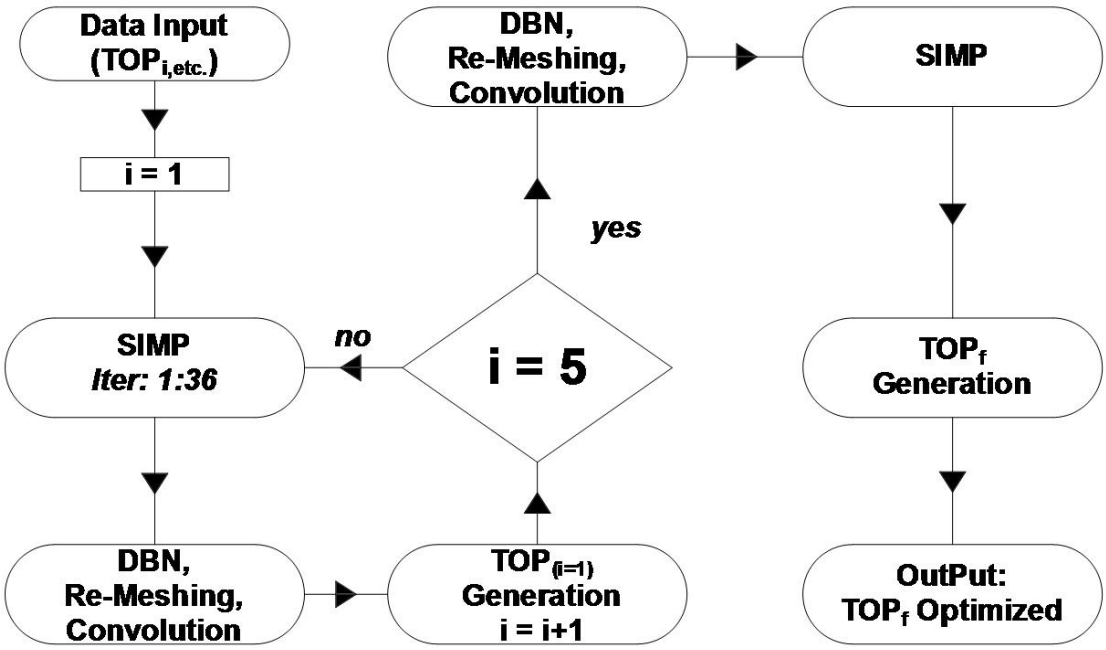
	$ne^{(0)}$			$ne^{(1)}$		
ne	406456	63480	32368	16200	8064	4000
$\frac{ne^{(0)}}{ne^{(1)}}$	1.00	6.40	12.56	25.09	50.40	101.61
$\frac{t^{(0)}}{t^{(1)}}$	1.00	12.89	27.72	59.35	137.00	374.70

Έστω ότι πρέπει να εκτελεστεί βελτιστοποίηση τοπολογίας στο πρόβλημα TOP_f. Σύμφωνα με τη μεθοδολογία DL-SCALE δημιουργούνται πέντε πανομοιότυπα μειωμένης τάξης μοντέλα TOP_i^R όπου η μόνη διαφορά με το αρχικό είναι το πλήθος των πεπερασμένων στοιχείων του πλέγματος.

$$\mathrm{TOP}_f:\ [L_{xyz}^{(f)}, ne^{(f)}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}] \longrightarrow \left\{ \begin{array}{l} TOP_1:\ [L_{xyz}^{(f)}, ne_1^{(\mathrm{R})}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}] \\ TOP_2:\ [L_{xyz}^{(f)}, ne_2^{(\mathrm{R})}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}] \\ TOP_3:\ [L_{xyz}^{(f)}, ne_3^{(\mathrm{R})}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}] \\ TOP_4:\ [L_{xyz}^{(f)}, ne_4^{(\mathrm{R})}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}] \\ TOP_5:\ [L_{xyz}^{(f)}, ne_5^{(\mathrm{R})}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}] \end{array} \right\}$$

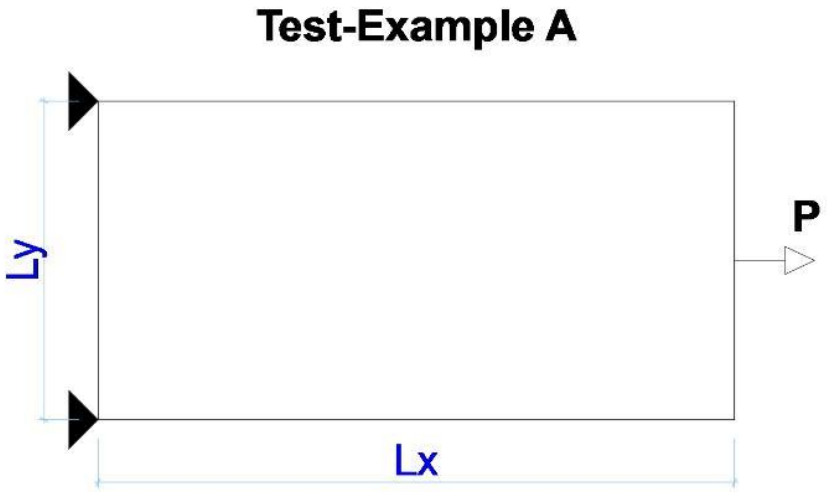
$$\text{where} \quad ne_1^{(\mathrm{R})} < ne_2^{(\mathrm{R})} < ne_3^{(\mathrm{R})} < ne_4^{(\mathrm{R})} < ne_5^{(\mathrm{R})} << ne_f^{(\mathrm{R})}$$

Μορφώνεται το πρόβλημα TOP_1^R και σε αυτό εφαρμόζεται η μεθοδολογία DL-TOP. Ως αποτέλεσμα αυτού προκύπτει η τελική τοπολογία του TOP_1^R. Στην τοπολογία αυτή εφαρμόζεται convolution και στο αποτέλεσμα που προκύπτει εφαρμόζεται πύκνωση του πλέγματος ώστε να φτάσει το πλήθος πεπερασμένων στοιχείων του TOP_2^R. Οι πυκνότητες του TOP_1^R μετά το convolution αντιστοιχίζονται με χωρικά κριτήρια στα πεπερασμένα στοιχεία του TOP_2^R και η παραχθείσα τοπολογία λαμβάνεται ως αρχική τοπολογία του TOP_2^R στο οποίο εφαρμόζεται το DL-TOP. Η διαδικασία αυτή επαναλαμβάνεται για κάθε TOP_i^R μέχρι η πύκνωση να φτάσει στο πλήθος πεπερασμένων στοιχείων του αρχικού μοντέλου TOP_0. Έχοντας πλέον υπολογίσει μια καλή εκτίμηση για το TOP_0, εκτελείται σε αυτό το DL-TOP και λαμβάνεται η τελική τοπολογία αυτού. Το διάγραμμα ροής του DL-SCALE αποτυπώνεται στην παρακάτω εικόνα:



Η μεθοδολογία DL-SCALE εφαρμόστηκε σε πλήθος προβλημάτων βελτιστοποίησης τοπολογίας ώστε να εκτιμηθεί η απόδοση και η σταθερότητά του. Ενδεικτικά αναφέρονται κάποια παραδείγματα ενώ τα υπόλοιπα συναντώνται στο τεύχος της διδακτορικής διατριβής.

Το πρώτο παράδειγμα περιγράφεται στην παρακάτω εικόνα.

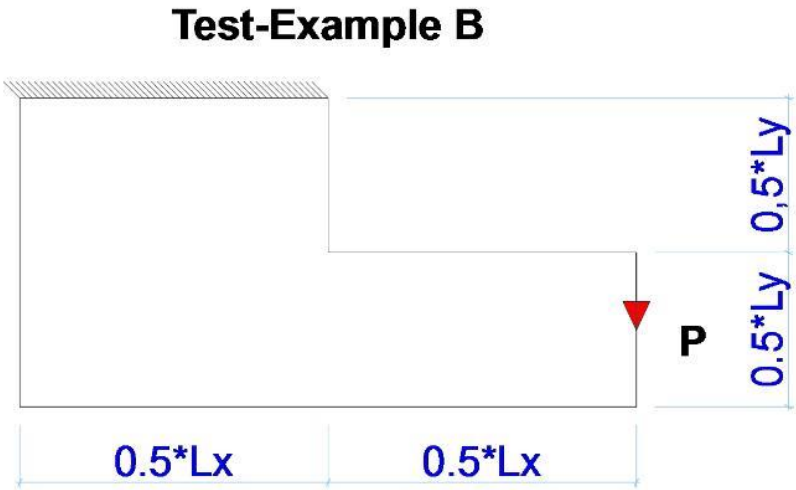


Οι διακριτοποιήσεις των TOP_i^R είναι 3.000, 4.000, 5.000, 7.000 και 10.000 πεπερασμένα στοιχεία και ο τελικός όγκος ίσος με 40% του όλου. Η μεθοδολογία δοκιμάστηκε σε τέσσερις διαφορετικές τελικές διακριτοποιήσεις ίσες με 20.000, 50.000, 75.000 και 100.000 πεπερασμένα στοιχεία. Η επιτάχυνση που επιτεύχθηκε παρουσιάζεται στον πίνακα που ακολουθεί.

ne	SIMP			DL-SCALE			Acceleration	Objective Function Value Reduction
	Iterations	Objective Function Value	Execution Time	Iterations	Objective Function Value	Execution Time	%	%
20,000	299	20.40	105.74	66	20.08	47.50	**55.08**	**1.54**
50,000	308	21.24	289.37	73	21.07	95.82	**66.89**	**0.82**
75,000	396	21.71	583.18	77	21.53	143.91	**75.32**	**0.85**
100,000	439	21.94	882.32	63	21.85	161.19	**81.73**	**0.40**

Οι τελικές διακριτοποιήσεις όπως προτάθηκαν από τη SIMP και την DL-SCALE είναι:

Το δεύτερο παράδειγμα περιγράφεται στην παρακάτω εικόνα.



Οι διακριτοποιήσεις των TOP_i^R είναι 1.000, 2.000, 3.000, 4.000 και 5.000 πεπερασμένα στοιχεία και ο τελικός όγκος ίσος με 35% του όλου. Η μεθοδολογία δοκιμάστηκε σε τέσσερις διαφορετικές τελικές διακριτοποιήσεις ίσες με 20.000, 50.000, 75.000 και 100.000 πεπερασμένα στοιχεία. Η επιτάχυνση που επιτεύχθηκε παρουσιάζεται στον πίνακα που ακολουθεί.

ne	SIMP			DL-SCALE			Acceleration	Objective Function Value Reduction
	Iterations	Objective Function Value	Execution Time	Iterations	Objective Function Value	Execution Time	%	%
20,000	220	144.36	68.03	65	141.31	33.23	**51.16**	**2.11**
50,000	322	139.14	259.31	60	136.73	62.32	**75.97**	**1.73**
75,000	403	139.46	499.28	59	137.41	88.45	**82.28**	**1.47**
100,000	375	138.16	631.16	57	136.86	113.85	**81.96**	**0.94**

Οι τελικές τοπολογίες που προτάθηκαν από τη SIMP και την DL-SCALE παρουσιάζονται στην παρακάτω εικόνα:

5.2 DLRM-TOP – Upscale μοντέλου μέσω deep learning

Η μεθοδολογία DLRM-TOP στοχεύει στην μείωση του απαιτούμενου υπολογιστικού χρόνου της βελτιστοποίησης τοπολογίας μέσω της χρήσης βαθιών νευρωνικών

δικτύων για την εκμετάλλευση πληροφορίας από τη βελτιστοποίηση αδρά διακριτοποιημένων μοντέλων.

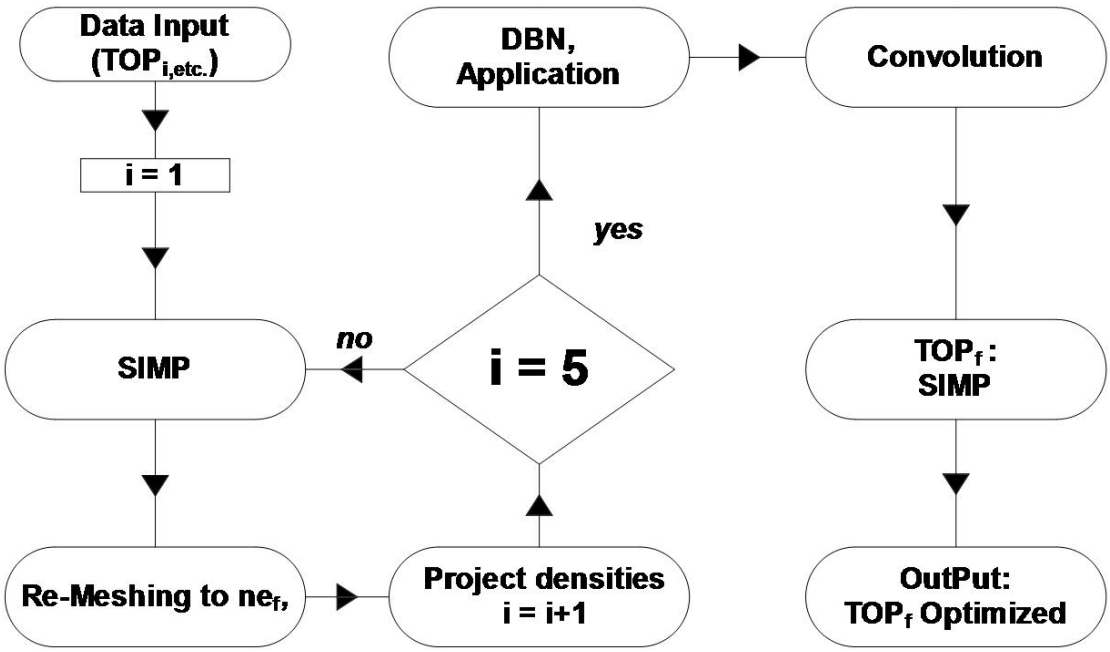
Σε αντιστοιχία με την μεθοδολογία του DL-SCALE, για την βελτιστοποίηση ενός μοντέλου, απαιτείται η δημιουργία πέντε όμοιων μοντέλων με μικρότερο πλήθος πεπερασμένων στοιχείων. Στην προκειμένη περίπτωση όμως κάθε TOP_i^R επιλύεται μέχρι τέλους με τη SIMP και δεν μεσολαβεί το DL-TOP. Οι τελικές τοπολογίες των επιμέρους TOP_i^R επαναδιακριτοποιούνται στο πλήθος πεπερασμένων στοιχείων του αρχικού μοντέλου και οι πυκνότητές τους προβάλλονται σε αυτό το πλήθος. Με τον τρόπο αυτό δημιουργείται για κάθε πεπερασμένο στοιχείο του αρχικού μοντέλου μια σειρά πέντε πυκνοτήτων που έχουν προκύψει από τα πέντε TOP_i^R.

$$D_{TOP} = \begin{bmatrix} T_{1,1}^{(d)} & T_{1,2}^{(d)} & \cdots & T_{1,ne_x^{(f)}}^{(d)} \\ T_{2,1}^{(d)} & T_{2,2}^{(d)} & \cdots & T_{2,ne_x^{(f)}}^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ T_{ne_y^{(f)},1}^{(d)} & T_{ne_y^{(f)},2}^{(d)} & \cdots & T_{ne_y^{(f)},ne_x^{(f)}}^{(d)} \end{bmatrix}$$

$$\text{where} \quad T_{i,j}^{(d)} = \left[\{d_{k,l}\}_{TOP_1}^{opt}, \{d_{m,n}\}_{TOP_2}^{opt}, \ldots, \{d_{b,c}\}_{TOP_5}^{opt} \right],$$
$$\text{and} \;\; i \in [1, ne_y^f],\; j \in [1, ne_x^f],\; k \in [1, ne_y^1],\; l \in [1, ne_x^1], \ldots, b \in [1, ne_y^5],\; c \in [1, ne_x^5] \qquad (1.20)$$

Αυτή η σειρά $T_{i,j}^{(d)}$ χρησιμοποιείται από το DBN για να προβλέψει την τελική πυκνότητα κάθε πεπερασμένου στοιχείου του αρχικού μοντέλου.

Το διάγραμμα ροής του DLRM-TOP παρουσιάζεται στην παρακάτω εικόνα:



Η χρήση του DLRM-TOP προϋποθέτει την εκπαίδευση του δικτύου μία φορά. Τα παραδείγματα στα οποία έγινε εκπαίδευση είναι όμοια με αυτά του DL-TOP μόνο

που τα δεδομένα εισόδου είναι διαφορετικά σε αυτήν την περίπτωση. Οι συσχετισμοί που χρησιμοποιήθηκαν καθώς και τα πλήθη των πεπερασμένων στοιχείων των μειωμένων (DAT) αλλά και του πλήρους μοντέλου (TOP_f) φαίνονται παρακάτω:

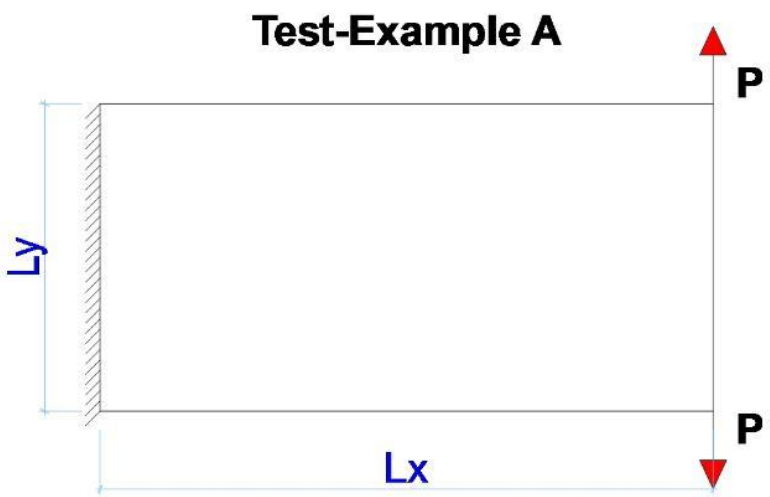
$$
\begin{array}{lcl}
\left[d_{TOP_1}^{opt}, d_{TOP_2}^{opt}, d_{TOP_3}^{opt}, d_{TOP_4}^{opt}, d_{TOP_5}^{opt}\right]^{(1)} & \rightarrow & d_{1,TOP_f} \\
\left[d_{TOP_1}^{opt}, d_{TOP_2}^{opt}, d_{TOP_3}^{opt}, d_{TOP_4}^{opt}, d_{TOP_5}^{opt}\right]^{(2)} & \rightarrow & d_{2,TOP_f} \\
\vdots \qquad\qquad \vdots \qquad\qquad \ddots & & \vdots \\
\left[d_{TOP_1}^{opt}, d_{TOP_2}^{opt}, d_{TOP_3}^{opt}, d_{TOP_4}^{opt}, d_{TOP_5}^{opt}\right]^{(ne-1)} & \rightarrow & d_{ne-1,TOP_f} \\
\left[d_{TOP_1}^{opt}, d_{TOP_2}^{opt}, d_{TOP_3}^{opt}, d_{TOP_4}^{opt}, d_{TOP_5}^{opt}\right]^{(ne)} & \rightarrow & d_{ne,TOP_f} \\
\text{Input} & & \text{Target}
\end{array}
$$

$$
DAT_i(\mathrm{ne}) = \begin{bmatrix}
1,000 & 2,000 & 3,000 & 4,000 & 5,000 \\
2,000 & 3,000 & 4,000 & 5,000 & 6,000 \\
4,000 & 5,000 & 6,000 & 7,000 & 8,000 \\
2,000 & 5,000 & 7,000 & 8,000 & 10,000 \\
6,000 & 7,000 & 8,000 & 9,000 & 10,000
\end{bmatrix}
$$

$$
TOP_f(\mathrm{ne}) = 200,000 \tag{1.21}
$$

Η μεθοδολογία DLRM-TOP δοκιμάστηκε σε διάφορα προβλήματα βελτιστοποίησης τοπολογίας όπου εξετάστηκε η απόδοσή της. Ενδεικτικά παρατίθενται τα αποτελέσματα από δύο από αυτά.

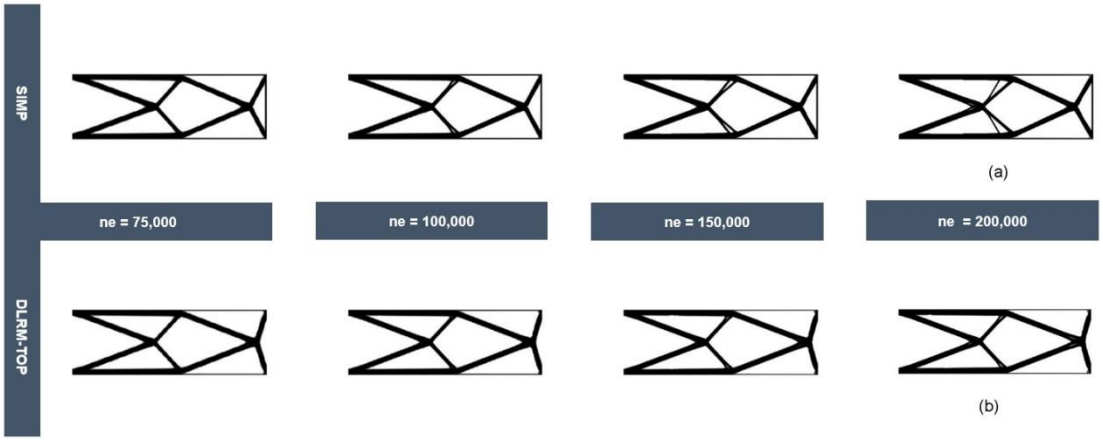
Το πρώτο παράδειγμα περιγράφεται στην παρακάτω εικόνα.



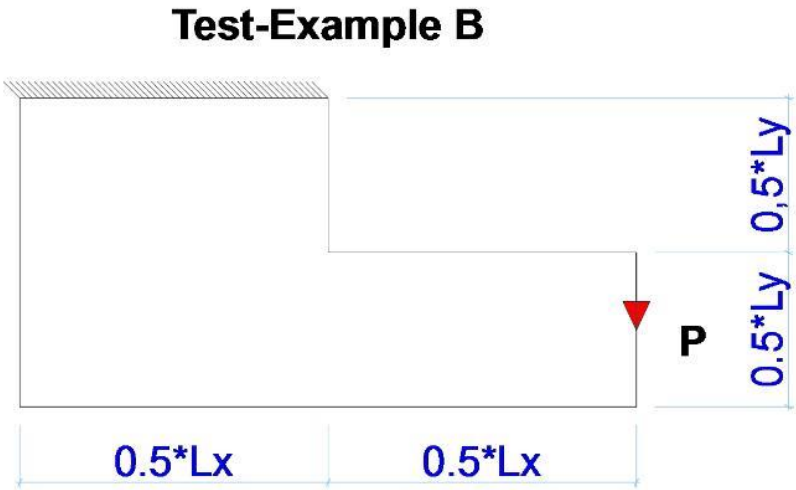
Οι διακριτοποιήσεις των TOP_i^R είναι 5.000, 7.000, 10.000, 15.000 και 20.000 πεπερασμένα στοιχεία και ο τελικός όγκος ίσος με 30% του όλου. Η μεθοδολογία δοκιμάστηκε σε τέσσερις διαφορετικές τελικές διακριτοποιήσεις ίσες με 75.000, 100.000, 150.000 και 200.000 πεπερασμένα στοιχεία. Η επιτάχυνση που επιτεύχθηκε παρουσιάζεται στον πίνακα που ακολουθεί.

ne	SIMP			DL-SCALE			Acceleration	Objective Function Value Reduction
	Iterations	Objective Function Value	Execution Time	Iterations	Objective Function Value	Execution Time	%	%
75,000	251	248.41	404.89	80	249.40	219.16	**45.87**	**-0.40**
100,000	331	246.46	736.53	61	247.37	224.06	**69.58**	**-0.37**
150,000	340	243.52	1203.95	72	245.15	337.66	**71.95**	**-0.67**
200,000	521	242.62	2559.05	86	244.39	495.55	**80.64**	**-0.73**

Οι τελικές τοπολογίες που προτάθηκαν από τη SIMP και την DLRM-TOP παρουσιάζονται στην παρακάτω εικόνα:



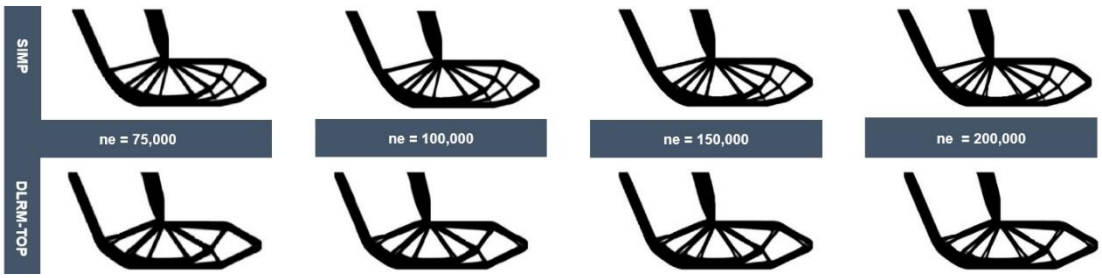
Το δεύτερο παράδειγμα περιγράφεται στην παρακάτω εικόνα.



Οι διακριτοποιήσεις των TOP_i^R είναι 5.000, 7.000, 10.000, 15.000 και 20.000 πεπερασμένα στοιχεία και ο τελικός όγκος ίσος με 30% του όλου. Η μεθοδολογία δοκιμάστηκε σε τέσσερις διαφορετικές τελικές διακριτοποιήσεις ίσες με 75.000, 100.000, 150.000 και 200.000 πεπερασμένα στοιχεία. Η επιτάχυνση που επιτεύχθηκε παρουσιάζεται στον πίνακα που ακολουθεί.

Ne	SIMP			DL-SCALE			Acceleration	Objective Function Value Reduction
	Iterations	Objective Function Value	Execution Time	Iterations	Objective Function Value	Execution Time	%	%
75,000	401	159.86	505.88	79	158.83	292.43	**42.19**	**0.64**
100,000	444	158.44	774.02	58	157.83	291.76	**62.31**	**0.38**
150,000	514	156.35	1438.26	98	155.73	461.50	**67.91**	**0.40**
200,000	367	156.01	1382.55	107	155.10	595.76	**56.91**	**0.58**

Οι τελικές τοπολογίες που προτάθηκαν από τη SIMP και την DLRM-TOP παρουσιάζονται στην παρακάτω εικόνα:

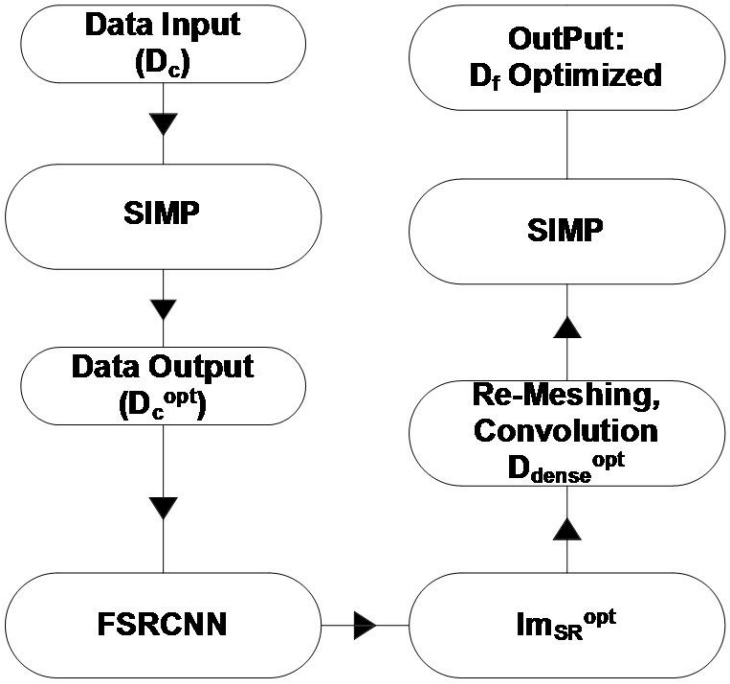


5.3 CN-TOP – Enhancing μοντέλου μέσω deep learning

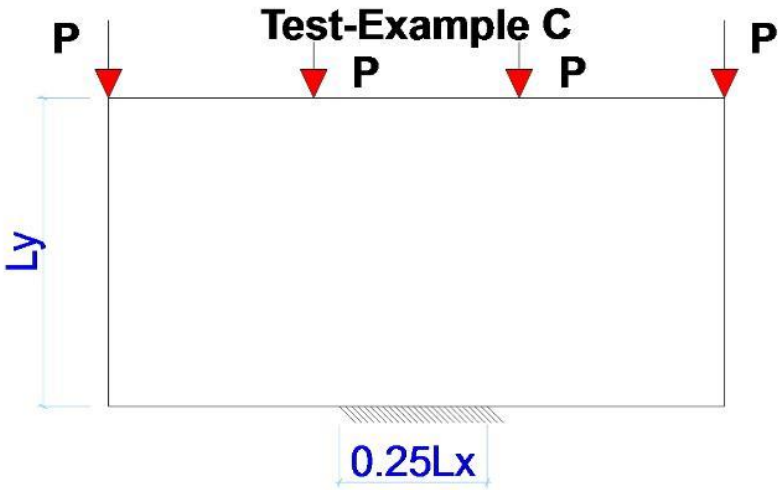
Η τρίτη μεθοδολογία που αναπτύχθηκε στα πλαίσια της παρούσας διδακτορικής διατριβής αφορά την εκμετάλλευση τεχνικών βελτίωσης της ανάλυσης της εικόνας

στην επιτάχυνση των διαδικασιών βελτιστοποίησης τοπολογίας. Αυτό μπορεί να επιτευχθεί μέσω της σύνδεσης της ποιότητας της εικόνας με την πυκνότητα του πλέγματος στην βελτιστοποίηση τοπολογίας. Πιο συγκεκριμένα, χρησιμοποιείται ένα δίκτυο FSRCNN [43] που έχει εκπαιδευτεί στην ανακάλυψη συσχετισμών ανάμεσα σε εικόνες από προβλήματα τοπολογίας που επιλύθηκαν με αδρή και με πυκνή διακριτοποίηση. Τα δεδομένα εκπαίδευσης του FSRCNN αποτελούνται από 10.000 ζεύγη βέλτιστων τοπολογιών που παράχθηκαν με την μέθοδο των Sosnovik και Oseledets [198] και τους Hunter et al. [84].

Σύμφωνα με την προτεινόμενη μεθοδολογία, για ένα πρόβλημα βελτιστοποίησης τοπολογίας δημιουργείται ένα όμοιό του αλλά με σαφώς μικρότερο πλήθος πεπερασμένων στοιχείων. Αυτό επιλύεται από την SIMP και το αποτέλεσμα της SIMP χρησιμοποιείται ως δεδομένο εισόδου του FSRCNN. Η έξοδος του FSRCNN είναι μια εικόνα σαφώς υψηλότερης ευκρίνειας, η οποία μετατρέπεται σε μητρώο πυκνοτήτων και επαναδιακριτοποιείται με τελικό πλήθος πεπερασμένων στοιχείων ίσο με το επιθυμητό. Αφού γίνει χωρική αντιστοίχηση πυκνοτήτων, το παραγόμενο μητρώο πυκνοτήτων εισάγεται στη SIMP και βελτιστοποιείται. Ένα διάγραμμα ροής προτεινόμενης μεθοδολογίας παρουσιάζεται στην παρακάτω εικόνα:



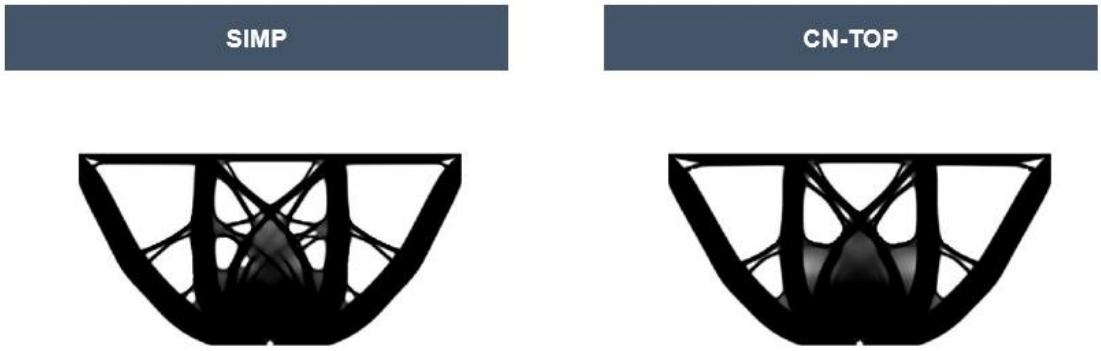
Η CN-TOP μεθοδολογία δοκιμάστηκε σε διάφορα προβλήματα βελτιστοποίησης τοπολογίας δύο διαστάσεων και τα αποτελέσματά της είναι ιδιαίτερα ενθαρρυντικά. Ενδεικτικά παρουσιάζεται ένα από αυτά ενώ τα υπόλοιπα περιλαμβάνονται στο τεύχος της παρούσας διδακτορικής διατριβής.
Το αρχικό πρόβλημα του παραδείγματος περιγράφεται στην εικόνα που ακολουθεί.



Το μειωμένης κλίμακας μοντέλο αποτελείται από 20.000 πεπερασμένα στοιχεία ενώ το πλήρες από 180.000. Η επιτάχυνση που επιτεύχθηκε παρουσιάζεται στον ακόλουθο πίνακα.

SIMP			CN-TOP			Acceleration	Objective Function Value Reduction
Iterations	Objective Function Value	Execution Time	Iterations	Objective Function Value	Execution Time	%	%
517	111.98	2243.77	142	112.19	634.77	71.71	-0.19

Τα σχήματα που προέκυψαν από τη SIMP και την CN-TOP παρουσιάζονται στην εικόνα που ακολουθεί.



6. Generative Design βασισμένο σε βαθιά νευρωνικά δίκτυα

Το Generative Design σκοπεύει στην γρήγορη παραγωγή από τον υπολογιστή πλήθους σχεδιασμών που σέβονται τους περιορισμούς που έχει θέσει ο χρήστης-σχεδιαστής χρησιμοποιώντας αλγορίθμους με επαναληπτικές διαδικασίες. Βασική του χρήση είναι η αντικατάσταση των περιορισμένου πλήθους αρχικών σχεδιασμών που μπορεί να δημιουργήσει ένας σχεδιαστής από πληθώρα αυτόματα παραχθέντων

σχεδιασμών [5,20,120,130]. Ως τεχνική χρησιμοποιείται πλέον τόσο στη βιομηχανία [7] όσο και στον κατασκευαστικό κλάδο [8,212]. Στο παρόν τμήμα της διδακτορικής διατριβής παρουσιάζεται μια μεθοδολογία Generative Design που αναπτύχθηκε στα πλαίσια αυτής και βασίζεται στα βαθιά νευρωνικά δίκτυα και την βελτιστοποίηση τοπολογίας. Πιο συγκεκριμένα, αποτελεί συνδυασμό των μεθόδων DL-SCALE, DLRM-TOP και SIMP. Η προτεινόμενη μεθοδολογία ονομάζεται DzAI$\mathbb{N}$.

Η εφαρμογή της μεθοδολογίας αυτής απαιτεί από τον χρήστη τον ορισμό του χωρίου μέσα στο οποίο θα παραχθεί ο σχεδιασμός, τις στηρίξεις αυτού, τις φορτίσεις αυτού και αν το επιθυμεί και τον τελικό στόχο όγκου εντός του χωρίου. Ακολούθως, ορίζεται η επιθυμητή διακριτοποίηση του πλέγματος και δημιουργούνται αυτόματα πέντε μοντέλα με μόνη διαφορά την σημαντικά πιο αραιή διακριτοποίηση. Καθένα από αυτά εισάγεται στην SIMP και εκτελούνται 35 επαναλήψεις. Καταγράφεται η πυκνότητα κάθε πεπερασμένου στοιχείου σε 6 διαφορετικά πλήθη επαναλήψεων. Κάθε στιγμιότυπο της βελτιστοποίησης στα 6 αυτά διαστήματα επαναδιακριτοποιείται με πλήθος πεπερασμένων στοιχείων ίσο με το τελικό επιθυμητό και προβάλλονται σε αυτό με χωρικά κριτήρια οι πυκνότητες κάθε πεπερασμένου στοιχείου. Με τον τρόπο αυτό παράγονται 6 διαφορετικά μητρώα όπου σε κάθε στοιχείο αυτών υπάρχει ένα διάνυσμα 5 πυκνοτήτων από τα 5 αραιά διακριτοποιημένα χωρία. Σε κάθε ένα από αυτά εφαρμόζεται η DLRM-TOP και στις 6 παραχθείσες τοπολογίες εφαρμόζονται 4 διαφορετικά convolution φίλτρα. Τα παραπάνω δύναται να συνοψιστούν ως εξής:

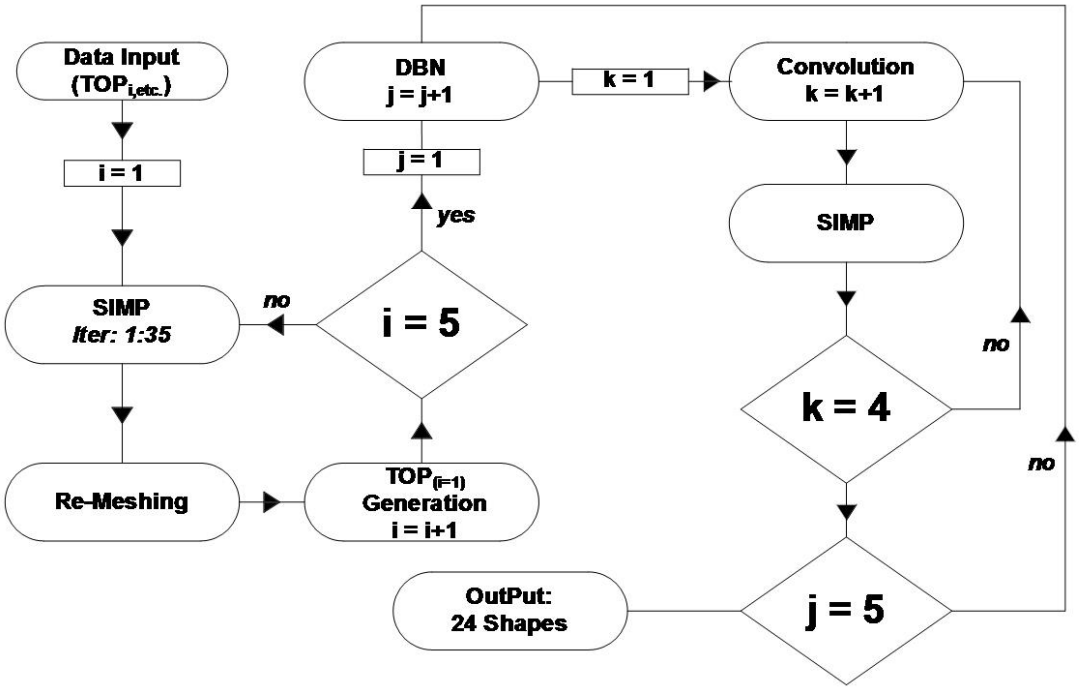
- $\mathrm{TOP}^{(\mathrm{f})}: \quad [L_{xyz}^{(\mathrm{f})}, ne^{(\mathrm{f})}, S_c^{(\mathrm{f})}, P_c^{(\mathrm{f})}, V_t^{(\mathrm{f})}]$
- Define TOP_i^R where $i \in [1,2,3,4,5]$
- Record D_j^i (extrapolated density matrix)
  where $j \in [5,10,15,20,25,35]$ (iterations counter)
- Create $T_z^k = \begin{bmatrix} T_{1,1}^{(d)} & T_{1,2}^{(d)} & \cdots & T_{1,ne_x^{(f)}}^{(d)} \\ T_{2,1}^{(d)} & T_{2,2}^{(d)} & \cdots & T_{2,ne_x^{(f)}}^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ T_{ne_y^{(f)},1}^{(d)} & T_{ne_y^{(f)},2}^{(d)} & \cdots & T_{ne_y^{(f)},ne_x^{(f)}}^{(d)} \end{bmatrix}, \; k \in [1,2,3,4,5,6]$

  $where \quad T_{i,j}^{(d)} = \left[\{d_{i,j}\}_{D_j^{i(F)}}^k, \{d_{i,j}\}_{D_j^{i(F)}}^k, \ldots, \{d_{i,j}\}_{D_j^{i(F)}}^k \right],$

  $and \;\; i \in [1, ne_y^f], \; j \in [1, ne_x^f]$
- Apply DBN to each of the 6 T_z and 4 different convolution filters in the proposed output.

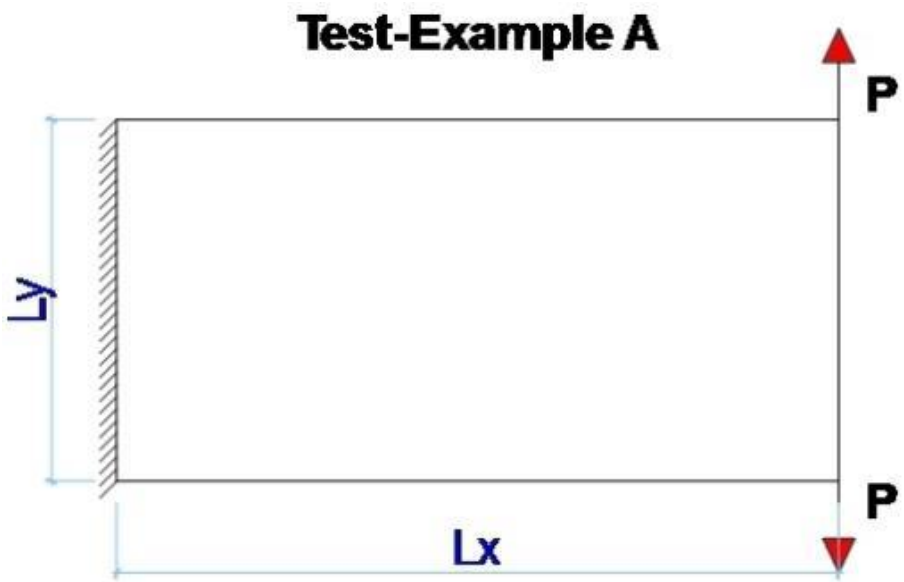
(1.22)

Αυτό έχει ως αποτέλεσμα την δημιουργία 24 τοπολογιών, οι οποίες εισάγονται στη SIMP για fine-tuning. Το διάγραμμα ροής της μεθοδολογίας DzAI$\mathbb{N}$ παρουσιάζεται στην παρακάτω εικόνα:

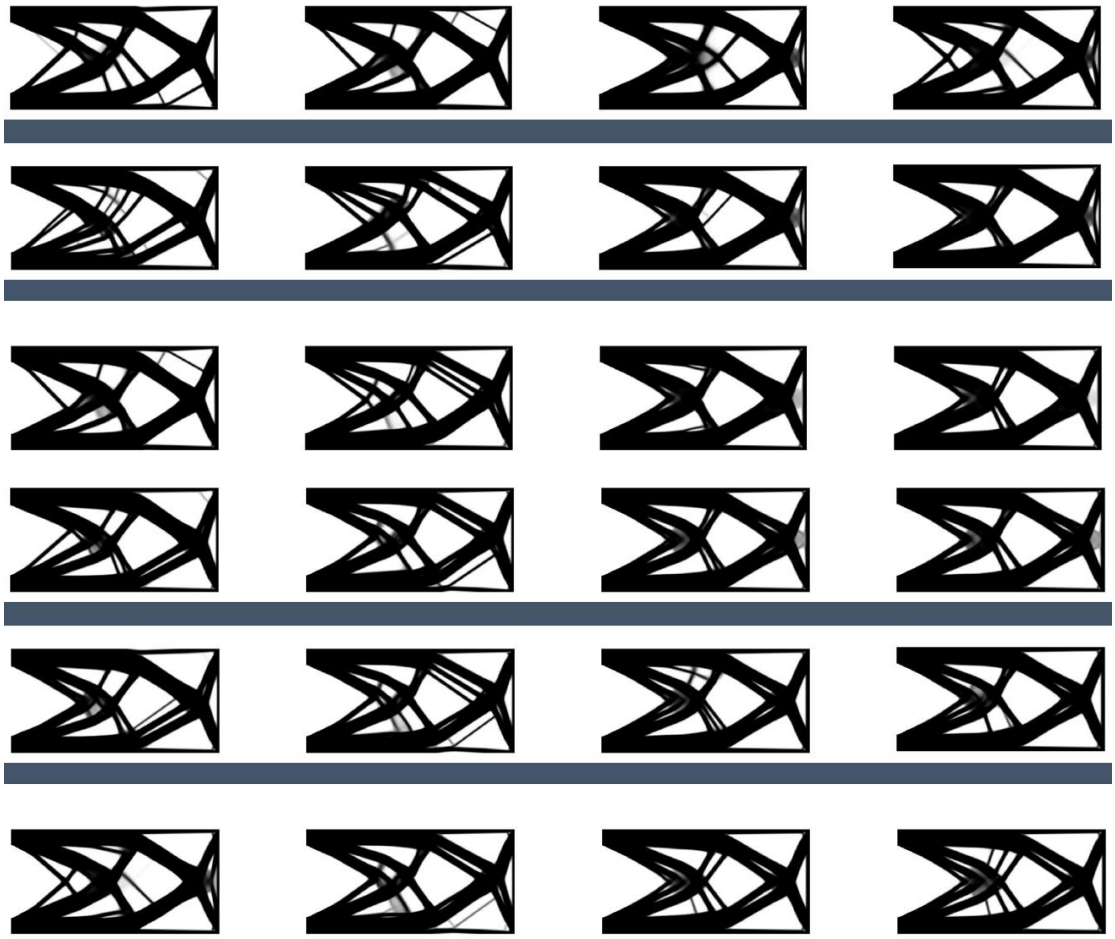


Η μεθοδολογία DzAIN δοκιμάστηκε σε σειρά από παραδείγματα εκ των οποίων παρουσιάζεται το ακόλουθο ενώ τα υπόλοιπα υπάρχουν στο τεύχος της παρούσας διδακτορικής διατριβής.

Το πρόβλημα του παραδείγματος αυτού ορίζεται όπως φαίνεται στην παρακάτω εικόνα:



Οι 24 παραχθέντες σχεδιασμοί όπως προέκυψαν από τη χρήση της μεθοδολογίας DzAIN παρουσιάζονται στην παρακάτω εικόνα:



Είναι ορατές οι διαφορές των προτεινόμενων σχεδιασμών όπως προέκυψαν από την DzAIN. Ο χρήστης-σχεδιαστής έχει τη δυνατότητα να επιλέξει όποιον ή όποιους σχεδιασμούς επιθυμεί ώστε να συνεχίσει την επεξεργασία και να καταλήξει στον επιθυμητό σχεδιασμό.

I would like to dedicate this thesis to the people who were there
for me during the past years . . . being patient with my absence

N. Ath. Kallioras

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Nikolaos Ath. Kallioras
April 2019

Acknowledgements

Most educational phases are characterized by a predefined, usually annual, goal. From a student's point of view it may be time-consuming, difficult and tiring but the course is set. The last of these phases is either the diploma or a master's degree. When deciding to pursue a Ph.D degree, there is no perception regarding the "education-wise" difference of this step. In order to broaden one's horizons, it is necessary for several mountain tops to be conquered as in all the previous educational goals. The main difference is that neither those tops nor the fields behind them are visible from the beginning. The path to a Ph.D is a journey on uncharted territories which are covered in local minima and maxima. Willingness to step out of the comfort zone and close to infinite persistence are mandatory as new capabilities must be developed in order to overcome the inevitable large number of failures. Despite the fact that it is a one-person course, the presence and assistance of key people is also of great importance.

Since I am approaching the end of my Ph.D, I feel the need to express my most sincere gratitude to my supervisor Professor Nikolaos D. Lagaros firstly for the inspiration, drive and guidance he provided. I also feel the need to thank Professor Lagaros for his continuous willingness to provide support through long lasting, mind-opening discussions both on the scientific field and every day issues as well. But most of all, I must express my appreciation for the countless hours of constructive collaboration on setting the path of my Ph.D course.

I also feel obliged to thank Professor Kyriakos Giannakoglou, as member of the Advisory Committee for the important and useful advices he provided on the various phases of my dissertation based on his broad field of expertise and the significant insights regarding the completion of my thesis.

A very significant part on the completion of my dissertation is ode to the useful comments and suggestions made by Prof. Vlasis Koumousis, Prof. Christos Provatidis, Prof. Evaggelos Plevris, Prof. Dimos Charmpis and Prof. Georgios Tsiatas as members of the Committee and I would like to express my gratitude to each one of them.

A sincere thank you must be expressed towards the School of Civil Engineering and its Personnel for providing the means, space and environment for performing my research and for hosting me from my graduate years and up to now.

I should also thank the other members of Prof. Lagaros' research team, Dr. Stavros Chatzieleftheriou and Ph.D candidates Mr. Stefanos Sotiropoulos, Mr. Giorgos Kazakis and Mr. Spyros Damikoukas for the harmonic and productive collaboration. Additionally, I would like to thank my friends for helping me with carrying on every day each one with his own way.

A special reference must be made to late Prof. Matthew G. Karlaftis, as due to him, my first approach on optimization and research was made. His scientific integrity but most of all his remarkable personality will always be remembered and act as guide for me.

My deepest gratitude is owed to my family, my parents Athanasios Kallioras and Evaggelia Kalliora, my wife Korina Gouvioti and to my two daughters, Athena and Evaggelia. Without their never-ending support, understanding and inspiration I would not have been able to reach up to this day.

Abstract

Even since the first structures where designed by human, finding the optimal design was a major problem. This need continued to increase in a geometrical manner as structural domain size and complexity also increased. Analysis of structures and its computational cost is also depended on the size and complexity of the structural system. The use of approximation methods increased due to the increase of computational cost of exact methods in analysis and design of structures. Additionally, a remarkable increase of available computational hardware was achieved with up to date computer processing units and the use of graphical processing units in calculations.

Some of the approximation methods used in the past where gradient-free algorithms, neural networks, reduced order models and combinations of them with each of the above presenting different advantages and disadvantages. Derivative-free algorithms have proven to be efficient in handling problems up to a number of parameters while interest on neural networks deprecated as training of large models was unsuccessful for quite some time. On the contrary, advances made in the last decade on training deep models have attracted interest on these methods, mainly on image and video analysis, natural language processing and big data fields. Surrogate and/or reduced order models are still used in several fields where analysis of large models is needed.

Work on gradient-free algorithms, surrogate modeling methods, machine learning algorithms and topology optimization is presented in this dissertation. Applications of the above methodologies and combinations of them in analysis and design of structures are also presented. In detail, a new, nature-inspired, metaheuristic algorithm is proposed and tested on NP-hard combinatorial problems while an improved version of Harmony Search Algorithm (HS) (firstly introduced by Zong Woo Geem) is also presented.

Additionally, the application of different deep neural networks on accelerating topology optimization procedure was examined and four different approaches are presented. The first one, DL-TOP uses a deep belief network in order to predict the final result of Solid Isotropic Material with Penalization (SIMP) method according to a few initial iterations. In a model upgrading approach, the second proposed solution DL-SCALE, focused on model upgrading, used a deep belief network to accelerate topology optimization methods by sequentially

providing close-to-final results from less dense discretizations to dense ones. The third proposed methodology, DLRM-TOP, again uses a deep belief network to predict the final result of a topology optimization method applied on a dense discretization by providing as input the results of significantly less dense discretizations. The final proposed methodology, CN-TOP, focuses on model enhancing by using a deep convolutional neural network to increase the density of a topology optimization result mesh in order to accelerate necessary computational time.

The formulated combinations of topology optimization method SIMP and deep learning methods proposed are modified and used as an AI-driven design of structures methodology, Dz**AI**$\mathbb{N}$. With Dz**AI**$\mathbb{N}$, a user can almost instantly produce a number of designs without any drawing input requested from his part, apart from the domain and loading and support conditions definition.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

MEM	PBA Memory
FE_{total}	Maximum Function Evaluations in PBA
TOP	Topology Optimization Problem
TOP_f	Topology Optimization Problem - final
$TOP_i^{(R)}$	i_{th} Reduced Topology Optimization Problem
ABC	Artificial Bee Colony Algorithm
ACO	Ant Colony Optimization
AI	Artificial Intelligence
ANN	Artificial Neural Network
BA	Bat Algorithm
BARON	Branch-And-Reduce Optimization Navigator
BBRBM	Bernoulli-Bernoulli Restricted Boltzmann Machine
BFGS	Broyden–Fletcher–Goldfarb–Shanno Method
BM	Boltzmann Machine
CB12	Cantilever Beam Database with 12 Classes in the OutPut
CB3	Cantilever Beam Database with 3 Classes in the OutPut
CD	Contrastive Divergence

CG	Conjugate Gradient Method
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CS	Cuckoo Search Algorithm
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DE	Differential Evolution
DFP	Davidon–Fletcher–Powell Method
DIRECT	DIvide a hypeRECTangle
DL	Deep Learning
ESO	Evolutionary Structural Optimization
FA	Firefly Algorithm
FE	Finite Element
FSRCNN	Fast Super-Resolution Convolutional Neural Network
GA	Genetic Algorithm
GBRBM	Gaussian-Bernoulli Restricted Boltzmann Machine
GD	Generative Design
GPU	Graphics Processing Unit
HMCR	Harmony Memory Consideration Rate
HMS	Harmony Memory Size
HS	Harmony Search Algorithm
IHS	Improved Harmony Search Algorithm
KH	Krill herd Algorithm
LHS	Latin Hypercube Sampling

MFO	Moth-Flame Optimization
ML	Machine Learning
MMA	Moving Asymptotes Method
OC	Optimality Criteria Algorithm
OIO	Optics Inspired Optimization
PAR	Pitch Adjustment Rate
PBA	Pity Beetle Algorithm
PCG	Preconditioned Conjugate Gradient Method
PS	Parameters Set of DBN training
PSO	Particle Swarm Optimization
RBM	Restricted Boltzmann Machine
RcNN	Recursive Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROM	Reduced Order Model
RST	Random Sampling Technique
SA	Simulated Annealing
SB12	Simply Supported Beam Database with 12 Classes in the OutPut
SB3	Simply Supported Beam Database with 3 Classes in the OutPut
SIMP	Solid Isotropic Material with Penalization
SRCNN	Super-Resolution Convolutional Neural Network
STO	Structural Topology Optimization
SVM	Support Vector Machine
TO	Topology Optimization
WWO	Water Wave Optimization

Chapter 1

Introduction

1.1 Dissertation scope

The main scope of the current dissertation is to contribute in the field of optimal structural analysis and design through developing and implementing new calculus methods by combining exact and approximation methods. As the size and complexity of analysis and design models is continuously increasing and the increase in the available computational power is not analogous, it is inevitable to investigate the exploitation of soft computing techniques. Exact methods have the ability to radically reduce the objective function of the problem in a relatively small number of iterations but lack the ability to overcome local minima. Actually, in real-life problems, it is more probable that the solution provided by exact methods will be a local minima than the global one. Additionally, the computational cost of calculating first and second-order derivatives or even the inability to calculate them resulted to examining the use of soft computing methods.

In a short historical review, it can be witnessed that the use of soft computing techniques grew dramatically from 1970 to 1990. Applications of such methods in real life problems were performed often and attracted significant research interest. Derivative-free methods, because of the randomness included, have the ability to overcome local minima and, practically, guaranty finding the global minima if there is no limitation to population of iterative solutions. The main drawback of these methods is that a significant amount of iterations is almost always necessary. Due to the fact that objective function calculation also became computationally heavy in the following years, the necessity of multiple calculations, inevitable in soft computing, was an important drawback. The above resulted to a drop of interest in soft computing as models became too complex for these methods.

In the last decade, research on modern soft computing methods drew a lot of attention, mainly because of the excessive amount of data generated, but not exploited, by companies

offering on-line services. That led to generating new methods, able to handle large amounts of data and extremely complex problem definitions. For example, deep neural networks have been a major scientific breakthrough in modern research.

The scope of this dissertation is to examine and apply modern soft computing methods in the fields of optimal analysis and design of structures. Some of the most computationally heavy problems dealt in structural design and analysis are the ones that include many solutions of the equation:

$$\{P\} = [K] * \{U\} \tag{1.1}$$

as inversing the stiffness matrix is rather time and computational resources consuming. To deal with this issue, techniques of reducing the size of the stiffness matrix, or minimizing the necessary iterative solutions' population or a combination of the previous two must be used. The core of this thesis is focused on providing new solutions in model order reduction and iterations population reduction in problems as the previously described ones. Additionally, parallel processing techniques are also examined in the solutions provided. Finally, a method for applying machine learning in the field of generative design is also presented.

1.2 Analysis of chapters

The current dissertation consists of seven chapters, the current one (Chapter 1 - Introduction) and six additional. In the second chapter, a description of the formulation of mathematical optimization is given along with a literature review of the state-of-the-art methods in optimization. Further along, a description of gradient-based algorithms is given along with detailed presentation of the two methods used in this dissertation, Optimality criteria algorithm (OC) [31] and Moving Asymptotes Method (MMA) [202]. Continuing, a thorough presentation of Improved Harmony Search algorithm (IHS) [94] can be seen. It describes a significant change that upgrades the performance of Harmony Search (HS), an algorithm presented by Zong Woo Geem [59].

In the same chapter, an algorithm proposed by Kallioras et al. [93], the Pity Beetle Algorithm (PBA) is also described. It is a swarm-type, nature-inspired, gradient-free algorithm. PBA was inspired by the aggregation behaviour, nesting and food searching patterns of the beetle named Pityogenes chalcographus (Six-toothed spruce bark beetle). This species detects and feeds off weakened trees inside forests. PBA is applied to several NP-hard single objective, unconstrained optimization problems of various scale, as it presents excellent results in handling large search spaces and finding the global minimum while avoiding

local minima. In order to investigate its performance, robustness and complexity, PBA was applied to a series of established benchmark, uni-modal and multi-modal, separable and non-separable test functions while results of the study are compared to state-of-the-art meta-heuristic algorithms. In Chapter 3, a thorough presentation of machine learning methods is recorded where the most established and modern are referenced. Additionally, a large part of Chapter 3 is devoted on up-to-date deep learning methods and specifically deep neural networks. In Chapter 3, an analytical description of Restricted Boltzmann Machines (RBM) [196, 71] is presented. RBMs are probabilistic graphical models, similar to Hopfield networks [54]. They can be represented as energy based twin-layer networks where all nodes of the first layer are symmetrically connected to all nodes of the other layer but no connections exist between nodes of the same layer[49]. Deep Boltzmann Machines (DBM) and Deep Belief Networks (DBN) are formed by sequentially connecting a population of RBMs. In this type of connection, the output of the previous, in line, RBM is also the input of the following RBM. Such deep models where successfully trained according to Hinton [75]. In this Chapter a description of Deep Convolutional Neural Networks (CNN) is also presented. CNNs are a type of deep neural networks inspired by the functionality of animal vision. They are designed in order to be able to accept as inputs 1D arrays, 2D matrices, or even 3D matrices [121]. Initially CNNs were used for image recognition or image classification [111]. From a networks architecture point of view, CNNs have hidden layers defined by their functionality (convolution, pooling, ReLU, normalization, etc.). After their successful performance in image recognition, CNNs were also applied on different problems such as natural language processing.

Chapter 4 is focused on proposing a technique for acceleration of topology optimization method and reducing computational cost of such method. The goal of topology optimization is to define the optimal performance-wise distribution of a specific volume of material inside a specific domain under specific loading and support conditions. One of the most knows topology optimization algorithms is Solid Isotropic Material with Penalization (SIMP) [12, 229, 147]. In an iterative manner, material volume that does not contribute significantly to the static system is eliminated. This procedure can be rather computational heavy in analogy to the finite element (FE) domain discretization and the necessary iterations. In this chapter of the dissertation, a method combining deep learning and SIMP is proposed in order to reduce computational cost. This method is called DL-TOP.

In DL-TOP, a DBN network is used in order to minimize necessary iterations of SIMP. In each element of the FE mesh a material density is calculated per SIMP iteration. This can be translated into a density time-history per iteration. A DBN is trained once on such time-histories aiming at reading correlations between a small population of initial time-steps

and the final density of time-history. This offers the possibility to perform a few iterations in a topology optimization problem and use the trained DBN to make a discrete jump from these iterations to a close-to-final solution, reducing dramatically the necessary iterations.

The main topic of Chapter 5 is acceleration of topology optimization methods and reduction of the computational cost of such methods via reduced order methodologies. Reduced order methods generated with the use of deep learning are proposed. In detail, three methods combining deep learning and SIMP are proposed in this dissertation in order to reduce computational cost. These methods are: DL-SCALE, DLRM-TOP and CN-TOP.

The first method, DL-SCALE, is focused on a model-upgrading approach with the use of a DBN alongside SIMP. In this method, initial SIMP steps and DBN predictions are used sequentially and in combination. This is performed iteratively starting from a much less dense discretization while increasing the discretization density per method step. With the use of DL-SCALE and sequentially upgrading the model, a significant drop of computational load is achieved. The second proposed method is DLRM-TOP, again from a model upgrading point of view. A DBN is used to predict the final output of a dense meshed test case based on results of a number of different, less dense discretizations, from small density to large. This data is used as a volume per discretization density for each finite element. This data is fed to a trained DBN and a close-to-final solution for a significantly more dense FE mesh. With this method, a large number of 'computationally expensive' iterations of SIMP are avoided. The fourth and final method proposed in this chapter is CN-TOP. CN-TOP is a model-enhancing application. A final output of SIMP with less dense mesh is used as input for a trained CNN. The CNN enhances the topology image resolution and the higher definition image is translated into a denser mesh. The enhanced output is then fine-tuned by SIMP. By using CN-TOP, a large number of SIMP iterations is avoided resulting to significantly less computational time.

In Chapter 6, work on Generative Design is presented. Firstly, a detailed presentation and review in up-to-date literature on generative design can be viewed. In generative design, the goal is to be able to produce almost instantly a number of designs of a structure formed by a computer. To achieve that, both computational methods like SIMP and machine learning applications like DBN are needed. A combination of the above are formulated into a generative design method in this dissertation, called Dz**AI**$\mathbb{N}$. In Dz**AI**$\mathbb{N}$, the designer must only define the domain and loading and support conditions as well as the non-optimized areas, if present. More parameters (i.e. the desired volume fraction) can be used as input but it is not mandatory. The use of SIMP and pretrained DBNs on the initial parameters defined concludes to several AI-created designs. The user can then select one or more of the designs as his inspirational or final model. Several AI-created models with the use of

DzAI$\mathbb{N}$ are presented in Chapter 6. In the final chapter, Chapter 7, a few thoughts regarding possible future work are noted. The experience and knowledge gained during the course of the PhD, has led to an increased interest on deep learning methods and applications in the analysis and design of structures. More applications of the above-mentioned techniques will be examined, as for example in mesh-generation and stiffness matrix reduction. Additionally, DzAI$\mathbb{N}$ will be further developed and improved in order to produce more results and take under consideration more design factors.

1.3 Contribution to international literature

The scientific contribution achieved during the current Ph.D. course can be categorized in four main fields: i) Metaheuristic algorithms, ii) Topology optimization acceleration, iii) Reduced order models, iv) Generative design

1.3.1 Metaheuristic algorithms

Contribution in the field of metaheuristic algorithms can be summarized in two main parts, the proposed improvement on Harmony Search algorithm and the proposed new metaheuristic Pity Beetle Algorithm. Regarding HS algorithm, a new version named IHS is proposed. The new formulation reduces complexity of the original one by reducing the possible algorithmic functions. This has a significant effect on the convergence speed, the computational load and robustness of the original algorithm. By reducing the algorithmic functions, a reduction of the parameters of HS is also achieved resulting to improved stability of the algorithm. IHS is tested on real-life problems and its performance is compared with HS and other well-established metaheuristics in terms of performance and consistency.

In the second part, a new metaheuristic algorithm called PBA is proposed. PBA is a nature-inspired, population-based, swarm-type algorithm. Inspiration came from the behavior of a beetle called Pityogenes chalcographus (Six-toothed spruce bark beetle). PBA has the ability to investigate large areas of possible solutions for the optimal one while avoiding entrapment in local optima. The complexity of the proposed algorithm is significantly small as most of the functions of PBA are similar and the choice of functionality depends on the quality of the solution generated at the previous position. The area exploration is performed with a "flight pattern" who's range is defined as described previously and position definition inside the search space range is defined with a Random Sampling Technique (RST) similar to Latin Hypercube Sampling (LHS) [152].

1.3.2 Topology optimization acceleration

Topology optimization is used by the designer engineer in order to shape the structural system that satisfies the predefined operating conditions is the best possible way. This is accomplished by eliminating parts of the initial domain that contribute insignificantly in the structural behaviour of the domain in an iterative manner. The computational demand of this procedure is quite high as it depends on the population of elements in the generated mesh. As a result, the necessary time for optimizing a fine meshed domain can even be equal to a number of days.

In the current thesis, a methodology called DL- TOP is proposed for minimizing the computational cost without any loss in terms of final result quality. DL-TOP is a combination of a topology optimization method (SIMP) and a deep learning method (DBN). In this novel two-phase methodology, the DBN is used for defining higher order correlations between the volume of an element in a number of initial SIMP iterations per finite element and the final density value of that element. The process begins with SIMP performing a few iterations which act as input for the DBN. Following that, the DBN proposes a value for each element in the mesh. The DBN proposal is then passed again to SIMP in order to perform a fine-tuning. The use of this method reduces computational cost in a range of 50 to 95% in all examples tested, both in 2D and 3D cases. In terms of objective function value (i.e. the compliance of the system), the values obtained present a difference with SIMP alone application in the range of [-1,1]%. It is worth mentioning that the DBN is trained once on two typical examples of topology optimization literature and applied on several examples without any additional training. Also the executional time of DBN is incomparably small in comparison with one single iteration of SIMP, especially in meshes of dense discretizations.

1.3.3 Model order reduction

Regarding contribution in model order reduction, three new methods based on deep learning are presented. These methods are DL-SCALE, DLRM-TOP and CN-TOP. The main idea behind these applications is to use deep learning in order to be able to extrapolate results of small scale models to larger ones without loss of quality.

As stated before, applying topology optimization on a domain with a dense mesh is a computationally heavy procedure. DL-SCALE is proposed as a model order reduction technique where instead of directly applying topology optimization on the dense-meshed domain, it is applied on more sparse-meshed ones. An identical domain in terms of dimensions, loading and support conditions, volume target, etc. but with a significantly sparser mesh is created as initial input of DL-SCALE. A few iterations t of SIMP are performed

and the DBN formulated for DL-TOP is used in order to propose a close to final solution. A more dense mesh is applied on the DBN-proposed shape where the volume of each new finite element is set equal to the volume of the closest one, in terms of coordinates from the previous, sparse-meshed, shape. The output is passed again into SIMP in order to perform another t iterations. In a similar manner, DBN is used to propose a result for the finner mesh and the same procedure continues until reaching a final shape with mesh density equal to the desired one.

As a result, by using DL-SCALE, SIMP is applied on sparse meshes before being applied on the desired one. Execution times of examples with sparse meshes are extremely smaller than the ones with a dense mesh. Additionally, when reaching the point of applying topology optimization in the target shape from mesh density point of view, the input proposed by DL-SCALE is close to a final one so only a few SIMP iterations are needed resulting to severely reduced computational time. Summarizing, the use of DL-SCALE allows the user to exploit results of reduced order models to avoid loss of time and severe computational loads of real-scale model optimization.

The second methodology proposed in this dissertation is DLRM-TOP. Starting with the desired mesh density of a topology optimization model, five models of sparser mesh density are created with identical structural and topology optimization parameters. Each finite element of the dense mesh is grouped with an element from each of the five sparse-mesh cases with respect to centers coordinates proximity. This leads to a creation of volume-density data for each element of the dense mesh. A DBN is trained on two typical, obtained from literature, examples where the input is the final shape proposed in five sparse meshed identical domains and the target of DBN training is the per element volume of an identical but dense meshed problem. The DBN is trained on the grouped volume/discretization per finite element data of two examples.

In order to use DLRM-TOP on a topology optimization problem, the user needs to apply SIMP on five identical but "sparsly" meshed problems. The final volumes per element per mesh are fed as input to the previously trained DBN and the network proposes an almost final volume value for each element of the dense meshed problem. This output is then fine-tuned by SIMP performing a small number of iterations. It is worth pointing out that the population of fine-tuning iterations are not user-defined but a result of the quality of the DBN output. The SIMP termination criterion is the same with typical SIMP approach. DLRM-TOP, as it can be witnessed on several test-cases presented in this dissertation achieves a significant reduction of computational time of topology optimization application by upgrading reduced order models without needing to retrain the DBN, regardless the optimization problem formulation.

The third and final methodology proposed in this section of the thesis with respect to reduced order modeling is CN-TOP. This methodology uses techniques developed for image handling in order to acquire information from reduced order models and use it to assist in handling large-scale models in topology optimization. In a 2D topology optimization problem, the output can be handled as a 2D image. Convolutional neural networks have been developed for enhancing image quality [42, 122, 105] while training CNNs on topology optimization results has also been proposed [198]. In this case, we train an image enhancing CNN with an input of small scale topology optimization results and an output of the same results but in larger scale.

As a result, SIMP can be applied on a reduced order model until termination criterion is reached. The output is read by the trained CNN which enhances the "resolution" of the SIMP results and provides a close to final result for the same problem but with a significantly finner mesh. This CNN output is fine-tuned by SIMP. As evident in all 2D test-cases examined, the application of CN-TOP in various topology optimization problems leads to severe reduction of execution time needed without any loss in terms of final result quality. Again, the population of fine-tuning iterations are not user-defined but a result of the quality of the CNN output.

1.3.4 Generative design

The strength of deep learning is the ability to produce a large amount of results in minimum time even without parallel processing. The time needed for deep neural networks to deliver an output since acquiring an input is in the scale of seconds even for large scale input and in serial execution. Another characteristic of deep neural networks is that networks trained on different examples can derive different results. The strength of combining a topology optimization method like SIMP and a deep learning method like DBN or CNN for reducing computational loads is thoroughly studied in the other parts of the dissertation. DBN is used successfully for making a discrete jump from an initial topology optimization schema to an almost final design. In a sense, it can be stated that the DBN forms a shape in accordance to an intuition received from SIMP. The final design is practically a computer generated design based on an input from SIMP.

The DBN proposed shape depends on the input given by SIMP, the database on which it was trained and the structure of the database. A large number of networks can be trained on varying databases as described before. This will result to networks with differentiating structures. Additionally, the population of SIMP initial iterations and different target volume percentages before the DBN is used can also produce variating inputs. Once a domain is defined along with support and loading conditions, Dz**AI**$\mathbb{N}$ is used to produce computer

generated designs. SIMP performs a series of iterations which are fed in parallel mode on several pre-trained DBNs. In minimum time, the DBNs generate forms that are returned to the designer as computer generated proposed shapes, fine-tuned by SIMP. It then up to the designer to select and work on one or some of the Dz**AI**$\mathbb{N}$ propositions.

1.4 Scientific work

During his Ph.D. course, the author of this thesis has contributed in the completion of 4 publications [91, 93–95] in scientific refereed journals, in writing 4 articles [87, 88, 90, 108] presented in international conferences and finally in the completion of two chapters [89, 92] in scientific books published by international publishers.

Chapter 2

Mathematical optimization and efficiency of novel algorithms

Optimization is the act of upgrading a certain solution with respect to specific criteria and under predefined limitations. From a structural design point of view, optimization is dealing with finding the optimal size, shape or topology of a structural system in terms of minimizing cost and/or maximizing performance goals while respecting set constraints. Nowadays, the main challenge in engineering is finding optimal solutions in modern, highly complex, formulations with the assist of advancements in available computational power. The optimality of the proposed solution is usually a combination of performance-cost criteria for the duration of the service life of the structure. Though engineering experience is key in assessment of the quality of each design, the complexity of todays structures imposes the use of numerical methods for automating the procedure of optimization. Numerous categorizations of methods for dealing with optimization problems can be found in literature. A more general one, is the separation in deterministic and stochastic ones according to their functionality. Another categorization is gradient-based and gradient-free methods according to the type of information needed to decide on the next search step. These methods are also separated into local search and global search ones depending on the type of the performed search. There are many more categories and sub-categories used but the previously mentioned are the general ones.These categories are explained more analytically in sections below.

2.1 Mathematical optimization formulation

A typical unconstrained optimization formulation can be represented as:

$$\begin{aligned} Minimize \quad F(X) \quad & where \\ & X = [x_1, x_2, ..., x_{n-1}, x_n] \\ with \quad respect \quad & to: \\ & x_i \in [x_i^{L_i}, x_i^{U_i}] \end{aligned} \tag{2.1}$$

where $F(X)$ is the objective function of the problem, X is the solution vector containing n variables and $x_i^{L_i}$, $x_i^{U_i}$ are the lower and upper bounds of the i_{th} variable. Additionally, the mathematical formulation of a constrained optimization problem can be represented as follows:

$$\begin{aligned} Minimize \quad F(X) \quad & where \\ & X = [x_1, x_2, ..., x_{n-1}, x_n] \\ with \quad respect \quad & to: \\ & g_k(X) \leq 0, \qquad k = 1, 2, ..., m-1, m \\ & l_j(X) = 0, \qquad j = 1, 2, ..., s-1, s \\ & x \in [x_i^{L_i}, x_i^{U_i}] \end{aligned} \tag{2.2}$$

where $F(X)$ is the objective function of the problem, X is the solution vector, $g(X)$ are the m inequality constraints, $l(X)$ are the s equality constraints, $x_i^{L_i}$, $x_i^{U_i}$ are the lower and upper bounds of the i_{th} variable.

From a historical point of view [170], optimization methods were initially introduced by Newton, Cauchy and Leibnitz. Additionally, Bernoulli, Euler, Lagrange and Weierstrass introduced calculus methods for unconstrained and constrained function minimization. After several years, the next advancement in optimization was by Dantzig and Bellman presenting the SIMPLEX method for linear programming [34] and the optimality principal [11] respectively. During the same period, Kuhn and Tucker [112] presented their contribution on the conditions for optimal solution as an extent to work presented by Karush [99]. Several more contributions were made in the fields of integer programming, stochastic programming and multi-objective optimization but interest was again raised during 2000 with plenty of work mainly on gradient-free optimization which will be presented in the following chapters.

2.2 Gradient-based algorithms

Gradient-based methods rely on first and/or second order derivative information for finding the optimal solution of optimization problems which usually is a local optima. The derivatives are used for defining the influence of variables to objective function values and finding the variable values that lead to optimal objective function value. The two key features of these type of algorithms are:

- Decide of the moving direction
- Decide the length of movement

Information acquired from calculation of derivatives is exploited for making the above decisions. The type of formulation used for taking advantage of derivative information is what usually differentiates gradient-based algorithms. Some of the most known algorithms of this type are:

- Steepest descent algorithm
- Conjugate gradient method
- Newton-Raphson method
- Quasi-Newton method

Cauchy introduced steepest descent algorithm [22] two centuries ago for finding the minimum of non-linear functions. It is based on the fact that a differentiable function reduces its value when moving stepwise in the direction where the gradient is negative [142]. When attempting to find the minimum of an objective function $f(x)$ where $x \in R^n$ is the variables vector, a start is made from an initial point x_i and the gradient $\nabla f(x_i)$ is calculated. The direction of movement is defined by $-\nabla f(x_i)$ while the new point x_{i+1} is given by $x_{i+1} = x_i + a_i * \nabla f(x_i)$ where $a_i = \arg\min_a f(x_i + a_{i-1} * \nabla f(x_i))$. This procedure continues in an iterative manner until a termination criterion is satisfied.

Conjugate gradient method (CG) [67], is actually based on gradient descent. As a type of line search algorithm, it performs iterative movements from one point to another according to information received from calculating gradient. The next point $(k+1)$ is calculated according to [64]:

$$x^{k+1} = x^k + a^k * d^k \tag{2.3}$$

where a^k is the size of the step of movement and d^k is the direction of movement. The step size is calculated via line search and d_k is calculated as follows:

$$d_{k+1} = -g_{k+1} + \beta_k * d_k \tag{2.4}$$

where $g_{k+1} = \nabla f(x_{k+1})^T$ and β_k is a parameter of the algorithm. As CG has proven to be very robust in handling optimization problems with large solution vectors, many variations of the algorithm have been proposed like Preconditioned conjugate gradient method (PCG) [55] which focuses on increasing convergence speed. The Newton-Raphson method is again an iterative method but second-order gradient information is needed. As all gradient-based algorithms, it performs a descent with direction and step information according to gradient at a specific point. The basic iteration expression for minimizing objective function $f(x)$, where $x \in R^n$ corresponds to the variables and starting from an initial point x_k, with this method is:

$$x^{k+1} = x^k - a^k * (\nabla^2 f(x^k))^{-1} * \nabla f(x^k) \tag{2.5}$$

where the direction is defined as:

$$d^k = -(\nabla^2 f(x^k))^{-1} * \nabla f(x^k) \tag{2.6}$$

where a^k is the size of the step to be performed [15]. While a local search method, some variations can be used for global search as well.

When applying the Quasi-Newton method in an optimization problem, the gradient $\nabla f(x)$ of the objective function $f(x)$ is used for generating estimators H_i for the Hessian matrix $H(x)$ of $f(x)$ at every iteration needed until converging [65] where:

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_{n-1}} & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_{n-1}} & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_{n-1}} & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.7}$$

Several variations of this method have been proposed in literature as it is a widely used algorithm in nonlinear optimization. Two of them are the DFP method [37, 52] and the BFGS method [19, 51, 62, 187].

2.2.1 Optimality Criteria Algorithm

Optimality criteria (OC) algorithm is commonly used for solving structural design optimization problems as it has proven to very successful against such problems [31]. In general, it can be stated that OC is based on the hypothesis that when in optimal position, an objective functions can be expressed by features that are not present in non-optimal positions. Description of the algorithm will be given based on application on such a problem and specifically, weight minimization of a structure S_t [148]. The general expression of the load-displacement relation is:

$$[K] * \{U\} = \{P\} \tag{2.8}$$

where U and P are the vectors of displacements and loads respectively and $[K]$ is the global stiffness matrix of the structure. The structure's weight can be calculated as follows:

$$W(S_t) = \sum_{i=1}^{m} \rho_i * A_i * l_i \tag{2.9}$$

where m is the number of finite elements in the structure, ρ_i is the mass density of the i_{th} element and A_i and l_i are the area and length of the i_{th} element respectively. The p constraints G of the above formulation could be generalized in the following expression:

$$G_j(A_i): \; C_j(A_i) \; - \; C_j^{Lower} \leq 0 \qquad \forall j = 1, \cdots, p \tag{2.10}$$

where C_j^{Lower} is the lower bound of the value of the j_{th} constraint. The Lagrange function of $W(A_i, \lambda_j)$, with the use of Eq. 2.9 and Eq. 2.10, can be expressed as follows:

$$W(A_i, \lambda_j) = \sum_{i=1}^{m} \rho_i * A_i * l_i + \sum_{j=1}^{p} \lambda_j * G_j(A_i) \tag{2.11}$$

where λ_j are the Lagrangian parameters. By differentiating Eq. 2.11, the optimal criteria assigned to the optimum are:

$$\rho_i * l_i + \sum_{j=1}^{p} \lambda_j * \frac{\partial G_j(A_i)}{\partial A_i} \; = \; 0 \quad \forall i = 1, \cdots, m \tag{2.12}$$

where $\lambda_j \geq 0$.

OC methodology, currently, is broadly used in the field of topology optimization as it will be described in following chapters.

2.2.2 Method of Moving Asymptotes

The method of moving asymptotes (MMA), introduced by K. Svanberg [202], is an iterative process applied in solving non-linear optimization problems with the use of a convex approximating subproblem.

MMA description

A general optimization problem with $f_0(X)$ being the objective function, can be defined as follows:

$$\begin{aligned} &Minimize \quad f_0(X) \quad for \quad X = [x_1, x_2, ..., x_{n-1}, x_n,] \\ &\quad subject \quad to: \\ &\qquad f_i(X) \leq \hat{f}_i, \qquad i = 1,2,...,m-1,m \\ &\qquad x_j \in [x_j^L,\, x_j^U] \qquad j = 1,2,...,n-1,n \end{aligned} \tag{2.13}$$

where $f_i(X) \leq \hat{f}_i$ express existing constraints while x_j^L and x_j^U are the lower and upper bounds of the j_{th} element of the solution vector X.

In the implementation of MMA, after an initial solution vector is chosen, iterations are performed for locating the optimal solution. For the solution vector X^k, created at the k_{th} iteration, the values of the objective function $f_0(X^k)$, the constraint functions $f_i(X^k)$ and their derivatives $\nabla f_i(X^k)$ are calculated. The first-order approximations are exploited for defining the solution vector of the $k^{th}+1$ step through application of duality method [115]. The value of $f_i(X^k)$ is calculated as follows:

$$f_i(X^k) = r_i^k + \sum_{j=1}^{n} \left(\frac{p_{ij}^k}{U_j^k - x_j} + \frac{q_{ij}^k}{x_j - L_j^k}\right) \tag{2.14}$$

where:

$$p_{ij}^k = \begin{cases} (U_j^k - x_j^k)^2 * \frac{\partial f_i}{\partial x_j}, & \text{if } \frac{\partial f_i}{\partial x_j} \geq 0 \\ 0 & \text{if } \frac{\partial f_i}{\partial x_j} \leq 0 \end{cases} \tag{2.15}$$

$$p_{ij}^k = \begin{cases} 0 & \text{if } \frac{\partial f_i}{\partial x_j} \leq 0 \\ -(x_j - L_j^k)^2 * \frac{\partial f_i}{\partial x_j}, & \text{if } \frac{\partial f_i}{\partial x_j} \geq 0 \end{cases} \tag{2.16}$$

and

$$r_i^k = f_i(X^k) - \sum_{j=1}^{n} \left(\frac{p_{ij}^k}{U_j^k - x_j} + \frac{q_{ij}^k}{x_j - L_j^k}\right) \tag{2.17}$$

As f_i^k is actually a first-order approximation of f_i at x^k, the second-order partial derivatives of f_i per x_j are:

$$\frac{\partial^2 f_i^k}{\partial {x_j}^2} = \frac{2 * p_{ij}^k}{(U_j^k - x_j)^3} + \frac{2 * q_{ij}^k}{(x_j - L_j^k)^3} \tag{2.18}$$

and:

$$\frac{\partial^2 f_i^k}{\partial x_j \partial x_i} = 0 \text{ if } j \neq 0 \tag{2.19}$$

resulting to:

$$\frac{\partial^2 f_i^k}{\partial {x_j}^2} = \begin{cases} \frac{2 * \frac{\partial f_i}{\partial x_j}}{U_j^k - x_j^k} & \text{if } \frac{\partial f_i}{\partial x_j} \geq 0 \\ \\ -\frac{2 * \frac{\partial f_i}{\partial x_j}}{x_j^k - L_j^k} & \text{if } \frac{\partial f_i}{\partial x_j} \leq 0 \end{cases} \tag{2.20}$$

It can easily be observed that the second order derivatives are positive while also, the closer the "moving asymptotes" U_j and L_j are to x_j, the larger the values of the second order partial derivatives are. According to the above, the subproblem is defined as:

$$\begin{aligned} & Minimize \quad \sum_{j=1}^{n} \left(\frac{p_{0j}^k}{U_j^k - x_j} + \frac{q_{0j}^k}{x_j - L_j^k}\right) + r_0^k \\ & subject\ to: \\ & \qquad f_i(X) \leq \hat{f}_i, \qquad i = 1, 2, ..., m-1, m \\ & \qquad x_j \in [x_j^L, x_j^U] \qquad j = 1, 2, ..., n-1, n \end{aligned} \tag{2.21}$$

2.3 Gradient-free algorithms

In several optimization problems, the calculation and/or use of derivatives is not possible due to several reasons. Either because derivative information is not available or it is too expensive to calculate or it is not reliable. Such cases are defined as gradient-free optimization problems and all algorithms used for dealing with these problems are known as gradient-free algorithms [173]. Ranging from structural engineering [109, 113, 114] to aerodynamics [98] or even currency portfolios [25] and medical science [136] gradient-free algorithms have a significantly broad spectrum of applicability.

From a historical point of view, gradient-free algorithms first appeared with Simplex algorithm [150] in 1965 but ever since, numerous algorithms have been presented incorporat-

ing different approaches of gradient-free optimization. In general, gradient-free algorithms can be divided in two major categories:

- Local search algorithms *subdivided into*

  Direct methods *and*

  Model-based methods

- Global search algorithms *subdivided into*

  Deterministic methods *and*

  Stochastic methods

with each category subdivided into separate types according to the method that describes their functionality.

Local search algorithms are characterized by moving from one identified solution to another one belonging in the same neighborhood while respecting well-defined rules [160]. A neighborhood is defined inside the search space X, $\forall\, x \in X$ and an initial position-solution $x_i \in X$ is chosen according to some rule. In an iterative manner, a new solution x_{n+1} is chosen from the neighborhood of x_n. The best solution found until termination criterion satisfied, is the local search optimization result.

Direct search method can be defined as sequential testing of trial solutions and comparison of each trial solution with the best one up to that moment in combination with a "next trial solution decision" strategy [78, 107]. Some of the most well-known methods of this category are Simplex [150], Generalized pattern search [107, 209] and Mesh adaptive direct search [6].

Model-based methods depend on information acquisition from surrogate models of higher quality. In a typical model-based method, a lower quality surrogate model is created at first while its quality is continuously increased through updating it according to new solution positions evaluation. Implicit filtering [61] and trust-region methods [165] are the most well-established model-based methods in literature.

Contrary to local search algorithms, the global search ones are not focused on exploring a specific area of the search space but are continuously exploring new areas of the search space while also perform "local searches" on the newly explored areas according to algorithmic rules. In order for the above feature to be implemented, most global search algorithms are population-based.

Deterministic global search algorithms apart from attempting to provide the global optima of a problem, they also focus on offering additive information regarding the found global optima, i.e. the bounds of the objective function on the global optima [53]. A few of the most

well established deterministic algorithms are Branch-And-Reduce Optimization Navigator (BARON) [178] and DIvide a hypeRECTangle (DIRECT) [86] algorithms.

Stochastic methods are used for solving global optimization problems by exploiting stochastic parameters either in the data of the problem or in the algorithmic implementation or even in both [228]. One category of such numerical methods are nature inspired stochastic global search algorithms which have proven to be capable of handling NP hard combinatorial problems by mimicking the evolution and survival-of-the-fittest procedure of species and specifically the sophisticated "survival" techniques developed by many species.

By studying these behaviors several algorithms referred to as heuristic and metaheuristic algorithms were inspired. Some modern and well established metaheuristic algorithms are: genetic algorithms (GA) [143], simulated annealing (SA) [211], particle swarm optimization (PSO) [102], differential evolution (DE) [36], harmony search (HS) [59], ant colony optimization (ACO) [44], artificial bee colony (ABC) algorithm [97], firefly algorithm (FA) [224], cuckoo search algorithm (CS) [224], bat algorithm (BA) [224], krill herd (KH) [57], variants of existing methods [183] and recently proposed ones, like improved artificial bee colony algorithm [191], water wave optimization (WWO) [227], the moth-flame optimization algorithm (MFO) [145] or the optics inspired optimization (OIO) [100].

Swarm optimization characterize a stochastic, population-based group of algorithms inspired by the social behaviour of birds flocking, fish schooling etc. [30]. Briefly, an initial population of particles (birds, fish, etc.) are positioned randomly into the multidimensional search space examined [30]. Every particle, whose position represents a solution, “travels” into the search space seeking for a better position/solution. During the iterations of the algorithm each particle adjusts its position based on its own experience, built by memorizing the best position encountered, as well as that of neighbouring particles. PSO algorithms combine local search (self-experience) with global search (neighbouring experience), aiming to balance exploration and exploitation. This procedure continues until the termination criterion is satisfied [125]. PSO algorithms have attracted a significant amount of interest in the past years [155, 210], since they were proved efficient in handling real-world optimization problems too [161].

2.3.1 Harmony Search and Improved Harmony Search Algorithms

Harmony Search algorithm (HS) [60] was inspired by the natural behavior of Jazz bands during an improvisation session. In the beginning of this procedure, one of the band members plays a randomly chosen note. The second band member, after listening to the first note played will decide to follow up with a second note. The decision on which this note will

be chosen is made either randomly or solely based on the musicians' musical memory of preferences or, finally, by applying a small change on a memory selected note. This procedure continues iteratively until a melody is composed. How this melody is received by the audience is the quality indicator of the improvisation outcome. In this thesis, an improved version of HS in terms of performance, robustness and computational needs, called IHS, is presented.

Mathematical formulation of HS

In an attempt to formulate HS in a mathematical manner, musicians are translated as decision variables, the pitch range of each musical instrument symbolizes the value range of each variable, the generated melody corresponds to a solution vector and finally, the acceptance of the audience can be translated as the objective function value [95]. The three main functions of HS algorithm are: harmony memory initialization, new harmony improvisation and harmony memory update.

In the *harmony memory initialization step*, the harmony memory (HM) is filled with a number of randomly generated solutions equal to the harmony memory size (HMS).

$$HMS = \begin{bmatrix} s_1^1 & s_2^1 & \cdots & s_{n-1}^1 & s_n^1 \\ s_1^2 & s_2^2 & \cdots & s_{n-1}^2 & s_n^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s_1^{HMS} & s_2^{HMS} & \cdots & s_{n-1}^{HMS} & s_n^{HMS} \end{bmatrix} \tag{2.22}$$

where n in the size of the solution vector.

In the *new harmony improvisation* step, the creation of a new solution vector is performed. The n elements of the vector are chosen based on three different functions:

- Random selection
- Memory consideration
- Pitch adjustment

In random selection, the value of the s_i variable is selected randomly with respect to lower and upper bounds. In memory consideration, the value of the s_i variable is chosen from a random position of the memory and finally in the pitch adjustment, the value of the s_i design variable is chosen by slightly modifying a value randomly selected from memory. The above described functionality can be summarized in [95]:

$$s_i^{New} = \begin{cases} s_i \in [x_i^{Lower}, x_i^{Upper}] & \text{with probability 1-HMCR} \\ s_i \in HM = [s_i^1, s_i^2, \cdots, s_i^{HMS}] & \text{with probability HMCR*(1-PAR)} \\ s_i + k & \text{with probability HMCR*PAR} \end{cases} \tag{2.23}$$

where HMCR (Harmony Memory Consideration Rate, $0 \leq HMCR \leq 1$) and PAR (Pitch Adjustment Rate, $0 \leq PAR \leq 1$) are parameters of the algorithm and k is a random number generated form a Poisson distribution.

Once the creation of a new melody - solution vector is completed, the algorithm moves to the *harmony memory update* step where the newly generated solution is compared against the worst one stored in HM with respect to objective function value. If it is better, it replaces it, else the algorithm rejects it and continues with iterations until a user-defined termination criterion is reached.

Mathematical formulation of IHS

It is of great importance for robustness of optimization algorithms to have the minimum possible number of parameters while it is also crucial in terms of complexity to have as few as necessary algorithmic functions. It is also important to reduce the necessary computational needs of an algorithm as much as possible. Under these principles, in this section, an improved Harmony Search algorithm (IHS) is proposed and tested against HS and other well established metaheuristics on two real-life problems.

In accordance with the three previously mentioned goals, there are two major differences in the mathematical formulation of HS and IHS [94]. The first difference is that instead of the three procedures for formulating a new solution vector in HS, only two are used in IHS. These procedures are *random selection* and *memory consideration*. The *pitch adjustment* procedure is incorporated into *memory consideration* in IHS. Due to this change, the parameter PAR is not needed in IHS formulation. As a result of the above, algorithmic complexity is reduced by one less function while also robustness is improved as there is one less parameter to tune. The second major difference is the way the solution vector is formed. Instead of using one of the algorithmic functions to propose an element of the vector, each function, solely, proposes a complete solution vector. Through this change, significantly less function calls are necessary per iteration while also, the efficiency of the algorithm is greatly improved as it can be seen in the testing of IHS in the "Application" part.

Accordingly, the mathematical formulation of new harmony generation can be seen in the following equation:

$$s^{New} = \begin{cases} \forall\, i \in [1,n] : \; s_i \in [x_i^{Lower}, x_i^{Upper}] & \text{with probability 1-HMCR} \\ \forall\, i \in [1,n] : \; s_i \in HM = [s_i^1, s_i^2, \cdots, s_i^{HMS}] & \text{with probability HMCR} \end{cases} \tag{2.24}$$

where n is the size of the solution vector.

2.3.2 Pity beetle algorithm

A new metaheuristic, swarm-type and population-based algorithm was developed during this Ph.D. course. Pity Beetle Algorithm (PBA), was inspired by the sophisticated technique that *Pityogenes chalcographus*, (six-toothed spruce bark beetle) (Coleoptera, Scolytinae), has developed for locating suitable nest hosts and food inside forests. Pity beetle is known for the ability to initially inhabit a single tree in a forest, spread rapidly by feeding of weakened trees and infest even the healthy trees once robust enough as a colony, a typical behavior of the Ipini tribe species [186].

In detail, a small number of male pioneer beetles, search the forest for suitable trees. Unhealthy trees are preferred for feeding of the bark and creating nests inside them. Once a weakened host is found, pheromone is spread from the beetles, in order to attract additional male and female beetles for the first brood to begin. Each of the males creates star-like nests inside the tree and in a polygamous manner, will mate with one to six females for the creation of the new generation. It is then time for the offspring to act as pioneer beetles and explore the forest for more suitable hosts. Once found, the previously described procedure is repeated for new brood generation. The possible traveling distance of the offspring depends on the quality of food in their birth position. Population outbreak is easily achieved if weakened trees are found in proximity. By translating forest into search space, trees as solution vectors and health of trees as quality of solution, it is noted that PBA is capable of handling large-scale optimization problems by efficiently searching the solution space in a sophisticated manner, avoiding local optima for global optima identification.

Further along, a sensitivity analysis with respect to algorithmic parameters is performed on PBA while also, the performance and robustness of PBA is evaluated and compared to other established, state-of-the-art metaheuristics with the use of several well-known benchmark uni-modal and multi-modal, separable and non-separable unconstrained functions. Further performance evaluation is executed with the use of the CEC 2014 unconstrained optimization benchmark test-suite [127] along with complexity testing.

Pityogenes chalcographus biology

The subfamily of Scolytinae contains a number of the most important forest pests in the world. Although various species of the subfamily present significant variations, all of them present an extremely sophisticated system for communicating with the use of pheromone signals. Bark beetles use such signals in order to succeed in mass population outbreaks which are a critical risk for the health of forests. Pityogenes chalcographus (Coleoptera, Scolytinae) is an important and rather common bark beetle in the area of Europe [106] as it usually attacks Norway spruce (Picea abies), pines (Pinus sp.) and larch (Larix decidua) [186, 163, 159, 220]. Other areas where the main host and in result Pityogenes chalcographus, is present are in central and northern Europe and it is also present in Elatia, Drama, Greece [9]. It is in most of its natural distribution bivoltine, producing two generations annually, depending however on the outside temperature [213]. When under excellent environmental conditions it can produce a third annual generation but when in higher altitudes, only one annual generation is possible [163, 151, 232, 231].

Pityogenes chalcographus is characterized by a polygamous behavior, where the male is mating with 3 to 6 females. The male P. chalcographus bore into the phloem of weakened trees by excavating a nuptial chamber. During their feeding procedure, pheromones are produced through transformation of host terpenes. These pheromones are used as female population attracting signals in order for the male and female beetles to mate in the nuptial chamber. From this star-like chamber, females start construction of mother galleries, depositing 40-70 eggs in egg niches [232]. Immediately after their hatching, larvae excavate galleries, horizontal to the mother galleries that end up in a pupal chamber where their development procedure is completed.

In an attempt to model the behavior of P. chalcographus, a series of specific stages can be defined. In the beginning, pioneer beetles search and find a suitable host (searching stage) by investigating the emission of chemical signals by weakened trees. Feeding on the bark of the host, an aggregation pheromone is created by pioneer beetles attracting other males and females (aggregation stage), increasing population in the area. As soon as a specific population is achieved, the defence mechanisms of the host can no longer resist this mass infestation, while at this population level, healthy trees can also be suitable hosts. It is also worth pointing out that an over-crowded host affects the infestation in a negative manner due to reduced feeding space and possible infectious diseases. To avoid such problems, when the population density in a host exceeds a certain upper bound, the beetles release an anti-aggregation pheromone that signals nearby beetles to not attack the specific tree but other trees in close distance (anti-aggregation stage). Through this process, the beetles

expand their territory in the forest by continuously creating groups of infested trees around the firstly colonized weakened tree.

Numerical implementation of PBA

The general form of an optimization problem can be expressed in standard mathematical terms as a non-linear programming problem as:

$$\begin{aligned} minF(x), x = [x_1, x_2, ..., x_D]^T \\ L_i \leq x_i \leq U_i, i = 1, 2, ..., D \end{aligned} \tag{2.25}$$

where $F(x) : R^D \rightarrow R$ is the real-valued objective function to be optimized (minimized in this case), $x \in R^D$ is the D-sized vector containing the design variables while the lower and upper bounds of the ith design variable are L_i and U_i respectively.
The main steps of the numerical simulation of the P. Chalcographus' remarkable host search and reproduction behavior, on which PBA is based, are described in detail. In the following presentation, with the term *population*, in PBA, the swarm of male and female beetles is described while each member of the beetles population is named particle. Additionally, the position vector of a beetle is presented as $x_j^{(g)} \in R^D$ where j is the ID of the population member, g is the generation (search step) and D is the dimension of the search space. PBA is characterized by three main steps: *Initialization, Host Selection Pattern* and *Update Location of Broods*. In the initial step of PBA, the first beetle brood is generated in a randomly selected position inside the search space (first generation). In the following step, members of the initial brood move to other host trees, in order to create the new broods (second generation). In every generation, new broods are created while in the third step new broods replace the previous ones. This procedure is repeated until a user-defined termination criterion such as maximum function evaluations (FE_{total}) or optimization goal (OpT_{target}) or executional time (ET_{max}) is satisfied. This procedure can be seen in Fig.2.1 in the form of a pseudocode, while in the next sections analytical descriptions of the algorithmic steps are presented.
where N_{broods} is the number of broods that are generated, N_{pop} is the population of pioneer beetles and FE_{total} are the maximum allowed function evaluations which is a termination criterion. Additionally, in Table 2.1 the range of values of the parameters of PBA are presented.

Random sampling - Hypervolume generation

The placement of pioneer beetles inside the flight-determined search area is performed with the use of a random sampling technique (RST). The key aspect of RST is that the

```
Begin
    g:=0
    Initialize() Eq. (6)
    Repeat
    For k := 1 to N_broods Do Begin
      For j := 1 to N_pop
        New Hypervolume Selection Pattern()
        Calculate F()
        FE := FE + 1
      End
    Update Population Position()
    End
    Until Termination Criterion (FE > FE_total)
End
```

Fig. 2.1 Pseudo-code of the pity beetle algorithm.

Parameter	Description	Value Range
N_{pop}	Population of pioneer beetles	[10, 100]
f_{nb}	Neighboring factor	[0.01, 0.20]
f_{tn}	Fine tuning factor	[0.005, 0.05]
f_{ms}	Mid-scale factor	[0.10, 1.00]
f_{ts}	Large-scale factor	[1, 100]
pr	Probability for choosing large-scale or memory consideration	[0, 1]
f_{FE}	Function evaluations multiplication factor	[0.05, 0.25]

Table 2.1 Algorithmic parameters of PBA and their range

N_{pop} discrete placement segments created properly represent the entire search space. In RST, a uniform distribution function for each variable is divided into a number of segments of equal marginal probability. The samples are defined by randomly selecting shuffled segments for each variable. To construct a sample, the range of each of the D variables is divided into N_{pop} non-overlapping equal segments, and then a sample with dimension D is created by randomly pairing the values of all parameters. In order to produce a sample of size D, a value is selected from each segment randomly. Following this procedure, N_{pop} samples are created [93]. Specifically in PBA, in order to randomly place N_{pop} particles

into a $D-sized$ space, the generation of D independent uniformly distributed variables $x_{i,j} \in [l_i, u_i], i = 1,2,...,D, j = 1,2,...,N_{pop}$ is necessary, resulting in $\prod_{j=1}^{D} N_{pop} = N_{pop}^{D}$ cells. The restriction that each hyper-row and column contain a single sample is satisfied by using the following expression for computing sample values:

$$x_{i,j} = F_x^{-1}\left(\frac{i-1+rx_i/(u_i-l_i)}{N_{pop}}\right) \tag{2.26}$$

where $rx_i \in [l_i, u_i]$ is a random number and F_x is the cumulative distribution function of the uniform distribution for $x_{i,j}$. The maximum combinations population of the sampling procedure for N_{pop} segments and D variables is:

$$(\prod_{d=0}^{N_{pop}-1} N_{pop}-d)^{D-1} = (N_{pop}!)^{D-1} \tag{2.27}$$

In each movement of each brood in PBA, a different search space length is used as it is analytically presented in the flight pattern descriptions. All variables are initially defined inside the search space described in Eq. 2.25 respecting the global lower and upper bounds. But in the progress of the algorithm, in each movement of a brood, different local bounds are used. For the gth movement, the lower bound $l_i^{(g)}$ and upper bound $u_i^{(g)}$ are used for calculating the i_{th} variable. The maximum flight (sample) distance $len_{max}^{(g)}$ is calculated accordingly:

$$len_{\max}^{(g)} = \left\| u^{(g)} - l^{(g)} \right\| = \sqrt{\sum_{i=1}^{D} \left(u_i^{(g)} - l_i^{(g)}\right)^2} \tag{2.28}$$

while the implementation of the sampling technique for the g_{th} sample will be defined as:

$$x_j = RST(l^{(g)}, u^{(g)}, D, N_{pop}) \tag{2.29}$$

Finally, after generating values for each variable in the range $[l_i^{(g)}, u_i^{(g)}]$, a check is performed to test whether global bounds are respected by the generated values, i.e $x_{i,j} \in [L_i, U_i]$. In case a violation occurs, a correction is performed where $\Delta len = u_i^{(g)} - l_i^{(g)}$ as it also described in Fig.2.2(a) and Fig.2.2(b).

Initialization step

In most of the well known and established population-based metaheuristics, it is usual to initialize the population via a random procedure [204] while this procedure has proven to be of significant importance in the overall performance of such algorithms [134]. Poor initial-

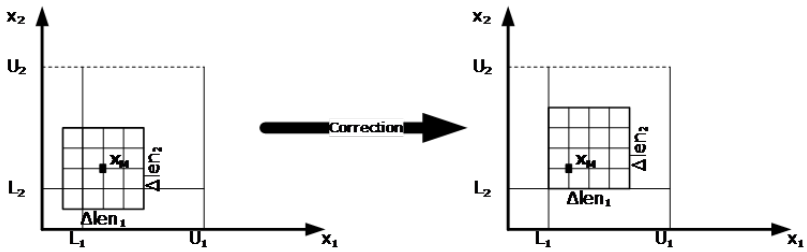


Fig. 2.2 Random sampling technique (a) before and (b) after correction.

ization usually results to early convergence to a solution of lower quality than when a more sophisticated initialization procedure is used. In order to not encounter early convergence problems, RST is used in the initialization step of PBA.

In the first step of PBA application, N_{pop} pioneer particles are positioned in a random manner inside the global search space with the use of RST, $x^{(0)} = [x_1^{(0)}, x_2^{(0)}, ..., x_{N_{pop}}^{(0)}]^T$, while the size of N_{pop} is a parameter of PBA. The generated hypervolume covers the entire search space and each positioned particle represents a solution vector for generation g=0 $x_j^{(0)} \in R^D$. The initialization step can be summarized as:

$$\begin{aligned} &x_j^{(0)} = RST(L,U,D,N_{pop}) \\ &\text{where} \\ &L = [L_1, L_2, \ldots, L_D],\ U = [U_1, U_2, \ldots, U_D] \text{ and } j = 1,2,\ldots,N_{pop} \end{aligned} \tag{2.30}$$

Once all pioneer particles have been placed in a position, the corresponding positions-solution vectors are compared with respect to objective function value. The particle placed in the optimal position releases pheromone to attract the rest particles to that position. This leads to establishing the first population. This step can be seen in Fig. 2.3(a) for N_{pop} equal to 9 and D equal to 2.

In the optimal position found so far, six different populations, each one having N_{pop} pioneer particles, will be created from six sequential births. A new flight-hypervolume size pattern is, then, be selected (one for each population of pioneer particles).

New Hypervolume selection pattern

Each group of new pioneer particles is planned to search inside the search space for other optimal solutions where new generations will be created. As previously mentioned, several types of flights (hypercube selection patterns) can be executed by pioneer particles with respect to hypervolume size. These selection patterns are:

- Neighboring search volume,

- Mid-scale search volume,
- Large-scale search volume,
- Global-scale search volume,
- Memory consideration search volume

A thorough presentation of the algorithmic implementation of each pattern is provided below. Regardless of the selection pattern used, a search space around the birth position is formed. The difference between each pattern lies on the area size which is controlled by PBA's parameter (f_{pat}). The pattern selection formulation for pioneer particles can be described as follows:

$$\begin{aligned} & x_j^{(g)} = RST(l^{(g)}, u^{(g)}, D, N_{pop}) \\ & \text{where} \\ & l_i^{(g)}, u_i^{(g)} \in \left[x_{birth,i}^{(g)} \cdot (1 - f_{pat}), x_{birth,i}^{(g)} \cdot (1 + f_{pat})\right] \end{aligned} \tag{2.31}$$

where i denotes the element component of the jth individual solution vector x_j, g denotes the generation (pursuit step), $x_{birth,i}^{(g)}$ is the ith element of the birth position/solution vector, $u_i^{(g)}$ and $l_i^{(g)}$ are the upper and lower bounds of the i_{th} dimension for the g_{th} pursuit step, respectively.

Neighboring search hypervolume

As in nature, it is certain that one or more of the generations created in a specific position will scavenge for a new suitable hosting position in close range to the birth location either because particles are not capable of covering a larger distance or there are plenty of suitable solutions at close distance or lastly due to over population in the birth position. Due to the previously described facts, the first hypervolume selection pattern is the *neighboring search* where a small hypervolume is defined around the initial generation position. The size of the neighboring search space is calculated with the use of the neighboring factor parameter, f_{nb}, whose value range is presented in Table 2.1. The hypervolume is defined with the use of Eq. 2.31 by setting $f_{pat} = f_{nb}$. Similarly to the initialization step, since all pioneer particles are set in a position, the one in the best position will attract the other ones but only if it is proven to be better than the starting position as well in terms of objective function value. The above can be described by Eq. 2.32:

$$x_{birth}^{(g+1)} = \left\{ \begin{array}{c} x_{birth}^{(g)}, \text{if} F(x_{birth}^{(g)}) < F(x_{j,k}^{(g)}), \forall j = 1,2,\ldots,N_{pop}, k = 1,2,\ldots,N_{broods} \\ x_{j,k}^{(g)}, \text{otherwise} \end{array} \right] \tag{2.32}$$

where g is the generation ID, $x_{birth}^{(g)}$ is the position vector and $x_{j,k}^{(g)}$ is the new position vector found in k population (k=1,2,...,N_{broods}). A 2D example for a neighbouring search hypervolume with $N_{pop} = 9$ can be seen in Fig. 2.3(b).

Mid-scale search hypervolume

In the case that *neighbouring search hypervolume* was unable to locate a position better than the birth one, repelling pheromones guide the particles to move to larger distances for an increased possibility of locating solutions of higher quality. In this pattern, the search area size is defined by the mid-scale factor, f_{ms}, parameter with a value range as proposed in Table 2.1. The hypervolume is created in accordance with Eq. 2.31 by setting $f_{pat} = f_{ms}$. The N_{pop} found positions are compared and the best one attracts the other pioneer particles for the creation of a new population. Again, an example of nine pioneer beetles performing a *mid-scale search* in a two-dimensional space is presented in Fig. 2.3(c).

Large-scale search hypervolume

Following the case that the *neighbouring search pattern* has not been able to locate a position more suitable than the birth one, the more robust members of the pioneer particles will move beyond the limits set in the *mid-scale pattern*. These particles will explore an even larger area, a large-scale hypervolume. This area is define with the use of the large-scale factor parameter of PBA, f_{ls} with value range as described in Table 2.1. As there is a possibility that the boundaries of the large-scale hypervolume will exceed the global ones, a check is performed and in such case, the violating boundary is replaced by the global one. In a typical manner, N_{pop} particles are placed inside the hypervolume and the one with the best objective function value attracts the rest for the start of new generations. The formulation of the *large-scale hypervolume* follows the method described in Eq. 2.31 by replacing f_{pat} by f_{ls}. A 2D image of this hypervolume for nine pioneer particles can be witnessed in Fig.2.3(c).

Global-scale search hypervolume

In the case that a large number of the previously described search types has failed in providing a better solution than the best found, repelling pheromones are released in all positions of generations. As a result, new pioneer particles are forced to perform a global search for an optimal position. The application of *Global scale search hypervolume* is imposed once a specific number of unsuccessful function evaluations, FE_{un}, is achieved. This number is a percentage of the total function evaluations, FE_{total}, allowed in the problem definition and is defined by $FE_{total} * f_{FE}$ where f_{FE} is a parameter of the algorithm with value range defined

in Table 2.1. The *global-scale search pattern* can be described as follows:

$$x_j^{(g)} = RST(L,U,D,N_{pop}) \tag{2.33}$$

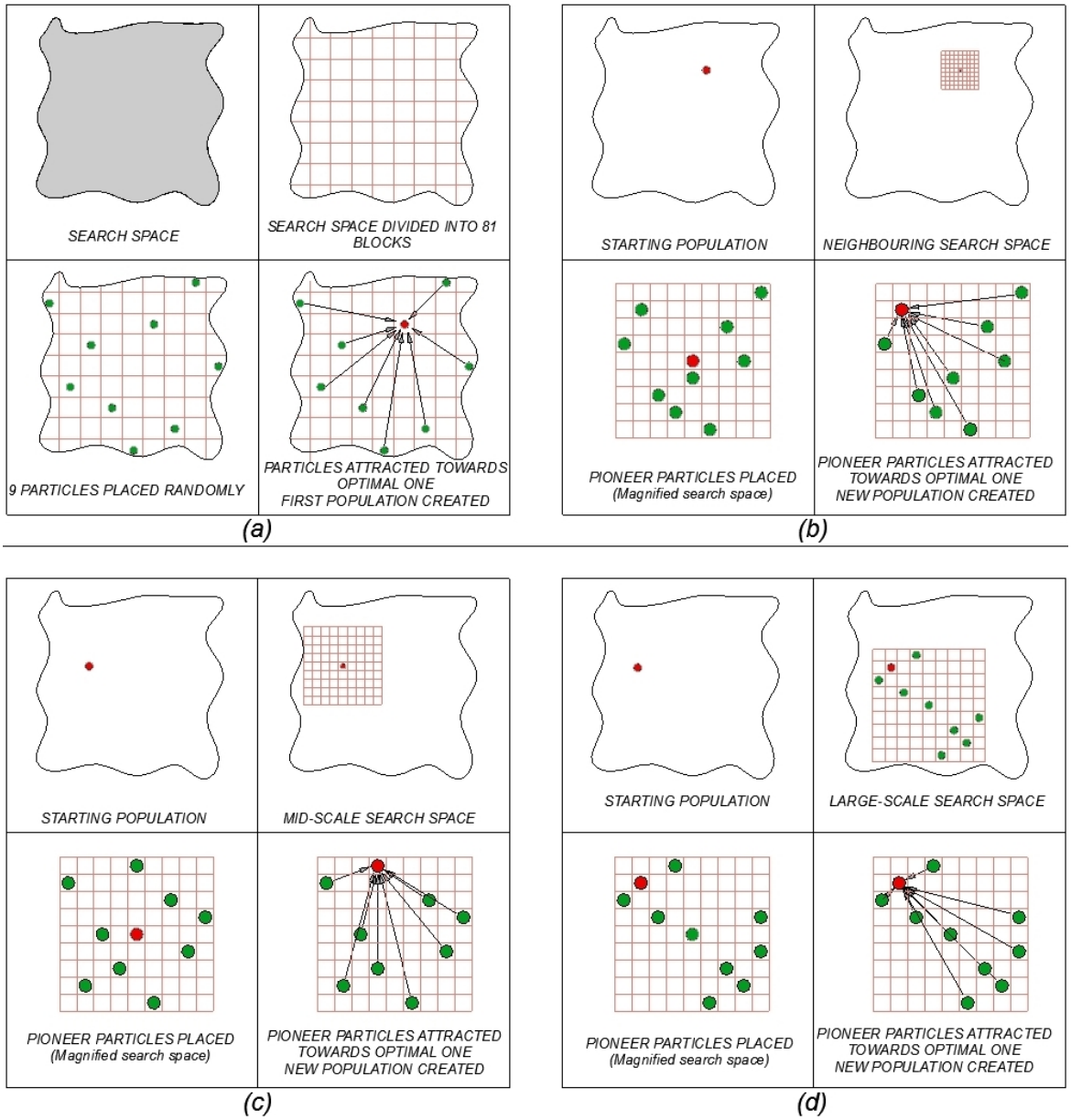


Fig. 2.3 Finding the new population position of pioneer particles - Hypervolume Selection Patterns. a) Initialization/Global-scale search hypervolume, b) Neighboring search hypervolume, c) Mid-scale search hypervolume, d) Large-scale search hypervolume

Memory consideration search hypervolume

According to the biological behavior of the beetle, weakened trees are the most suitable hosts except for the case that overpopulation is achieved, giving the ability of particles to attack healthy trees as well. This ability can be witnessed in cases where a large number of suitable hosts is present in close distance as pheromones will attract a large number of particles in the specific area. By having the ability to attack all possible hosts in proximity, the generations move by following a spherical expansion rule. To model this behavior, a population of N_{pop} best solutions found previously are stored in the algorithmic memory **MEM** and are subsequently used in order to produce new positions. It is worth mentioning that initially, the **MEM** is filled with the solutions found in the initialization step:

$$MEM = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,N_{pop}} \\ x_{2,1} & x_{2,2} & \dots & x_{2,N_{pop}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{D,1} & x_{D,2} & \dots & x_{D,N_{pop}} \end{bmatrix} \tag{2.34}$$

In the application of this pattern, a search space is created around each solution vector stored in **MEM** by this procedure:

- choose a **MEM** member
- $\forall i = 1,2,\dots,D$, changes in the range $[Li, Ui]$ are implemented while others are kept fixed
- if new found position is better than the worst in **MEM**, it replaces it
- a local search is performed around the best position found

The local search is performed according to Eq. 2.31 by setting $f_{pat} = f_{tn}$ where f_{tn} is a parameter of PBA with value range defined in Table 2.1.

Update location of populations

Once new positions are found for all generations of a brood, the birth locations are updated. This means that past birth places become obsolete and replaced by new ones. The only ones that remain are the solutions stored in **MEM**. Additionally, hypervolume search patterns are chosen for the newborn generations according to the previously described patterns except if the termination criterion of PBA execution is satisfied. The formulation of all hypervolume

patterns selection procedure is presented below in Eq. 2.35 and in Fig. 2.4:

$$x_{j,k}^{(g)} = \begin{cases} RST(\left[x_{birth,i}^{(g)} \cdot (1-f_{nb}), x_{birth,i}^{(g)} \cdot (1+f_{nb})\right], D, N_{pop}), if\ k=1 \\ else \begin{cases} RST(L,U,D,N_{pop}), if FE > FE_{un} \\ else \begin{cases} RST(\left[x_{birth,i}^{(g)} \cdot (1-f_{ms}), x_{birth,i}^{(g)} \cdot (1+f_{ms})\right], D, N_{pop}), \exists j, f(x_{j,k-1}^{(g)}) < f(x_{birth}^{(g)}) \\ else \begin{cases} RST(\left[x_{birth,i}^{(g)} \cdot (1-f_{ls}), x_{birth,i}^{(g)} \cdot (1+f_{ls})\right], D, N_{pop}), \text{if } r < pr \\ MEM, otherwise \end{cases} \end{cases} \end{cases} \end{cases}$$

$$i = 1,2,\ldots,D, j = 1,2,\ldots,N_{pop} \tag{2.35}$$

where r is a randomly generated number, $r \in (0,1)$. In Fig. 2.4, the decision routine for members of the k_{th} population is presented. In detail, the initial population is created with the neighboring search pattern in *line 2* with respect to the starting position quality. The remaining, up to N_{broods}, populations are sequentially created according to *lines 4 to 19*. A global scale search is performed if the threshold of maximum unsuccessful function evaluations, FE_{un}, is reached in accordance to *line 6*. If not, and in case the objective function value of the previous population is better than the one of the birth position, a mid-scale pattern is applied for this generation *line10*. If both the above criteria are not met, either a large-scale pattern *line 13* will be applied or memory consideration *line 15* according to a random rule.

2.4 Applications

In this section, applications of the two proposed algorithms (IHS and PBA) are presented. In detail, IHS is applied on a districting problem for optimal infrastructure inspection after a seismic event while PBA is applied on known-to-literature tests for new proposed algorithms.

2.4.1 PBA performance

In this section, an analytical presentation on the sensitivity analysis of PBA regarding the algorithmic parameters can be viewed along with a thorough examination and comparison of the performance of PBA against well established, state-of-the-art metaheuristics.

Sensitivity analysis and algorithmic parameters values

It is of great importance to properly define the parameters values of metaheuristics as the performance of these algorithms is strongly tied with those values. Algorithmic robustness is defined by the sensitivity of an algorithm with respect to parameter values. As there are no "one size fits all" solutions in such problems, a sensitivity analysis is used in order to define

1. For $k := 1$ *to* N_{broods} *Do Begin*
2. If $k == 1$ *Then*
3. $x_{j,k}^{(g)} = RST(\left[x_{birth,i}^{(g)} \cdot (1 - f_{nb}), x_{birth,i}^{(g)} \cdot (1 + f_{nb})\right], D, N_{pop})$
4. Else
5. If $FE > FE_{un}$ Then
6. $x_{j,k}^{(g)} = RST(L, U, D, N_{pop})$
7. Else
8. If $f(x_{j,k-1}^{(g)}) < f(x_{birth}^{(g)})$ Then
9. $x_{birth}^{(g)} := x_{j,k-1}^{(g)}$
10. $x_{j,k}^{(g)} = RST(\left[x_{birth,i}^{(g)} \cdot (1 - f_{ms}), x_{birth,i}^{(g)} \cdot (1 + f_{ms})\right], D, N_{pop})$
11. Else
12. If $r < pr$ Then
13. $x_{j,k}^{(g)} = RST(\left[x_{birth,i}^{(g)} \cdot (1 - f_{ls}), x_{birth,i}^{(g)} \cdot (1 + f_{ls})\right], D, N_{pop})$
14. Else
15. $x_{j,k}^{(g)} = MEM$
16. EndIf
17. EndIf
18. EndIf
19. EndIf
20. End

Fig. 2.4 Update location of populations.

ranges of values for which an optimization algorithm performs well on several different problems. The procedure followed for finding proper parameter value ranges is the following: Six test-function are used and 25 sets of combinations of parameter values are created with the use of random sampling technique. A total of 100 independent runs per test function and combination of parameter values are performed while best and worst performance with respect to objective function value, mean objective function value and standard deviation are calculated. This is necessary as since non-deterministic optimization algorithms do not yield the same results when restarted with the same parameters [172]. The combinations of parameter values used can be seen in Table 2.2.

Table 2.2 Samples of the algorithmic parameter values used in the sensitivity analysis

Set	***Npop***	***fnb***	***ftn***	***fms***	***fls***	***pr***	***fFE***
1	38	0.075705	0.029182	0.563642	57	0.338595	0.161134
2	24	0.123459	0.037083	0.624188	22	0.408397	0.21086
3	16	0.068691	0.007445	0.999846	34	0.692002	0.24552
4	47	0.045784	0.032992	0.935893	40	0.149911	0.091139
5	46	0.033457	0.045939	0.436588	5	0.905509	0.113425
6	26	0.017896	0.0306	0.81412	6	0.837562	0.214723
7	92	0.134056	0.037759	0.950333	43	0.965924	0.178206
8	52	0.083214	0.017674	0.899109	96	0.340249	0.176398
9	12	0.052221	0.026661	0.274668	18	0.710718	0.187977
10	61	0.195374	0.009146	0.531219	38	0.384507	0.23957
11	21	0.090543	0.020727	0.417393	66	0.008013	0.172095
12	29	0.162232	0.04883	0.493499	49	0.435726	0.079244
13	67	0.178636	0.046596	0.97612	87	0.055815	0.144725
14	65	0.063162	0.034171	0.858121	29	0.196453	0.169105
15	35	0.080927	0.042115	0.511393	84	0.581464	0.134154
16	36	0.031623	0.031709	0.829211	37	0.225093	0.11579
17	45	0.015055	0.049438	0.72741	3	0.502583	0.06186
18	96	0.113653	0.014282	0.862913	69	0.080354	0.202934
19	28	0.070926	0.006222	0.5524	58	0.530035	0.195361
20	32	0.136118	0.043162	0.171538	24	0.128949	0.071422
21	12	0.095019	0.01706	0.686054	7	0.271986	0.124869
22	58	0.087889	0.045057	0.910344	9	0.473968	0.218572
23	86	0.130502	0.010934	0.754645	33	0.620001	0.235528
24	62	0.039123	0.019051	0.78291	86	0.246476	0.056075
25	16	0.125928	0.024675	0.294616	59	0.876725	0.201997

The objective functions ($f_1(x)\ to\ f_6(x)$) used in the sensitivity analysis runs are presented in Table 2.3, along with the objective functions that complete the set used in the comparative study ($f_7(x)\ to\ f_{13}(x)$) presented in the next section. The dimension D of the test functions considered in the sensitivity analysis study is equal to 10 and 30 while the maximum function evaluations permitted for defining convergence were set to 100,000 and 200,000 for the two dimensions, respectively. For each run, record is kept for objective function value and the minimum and maximum objective function value, average value and standard deviation are calculated for each test-function and parameter set. The results obtained from the sensitivity analysis are presented in Tables 2.4 and 2.5, for $10-D$ and $30-D$ problems, respectively. In an attempt to summarize these results, it is worth noting that PBA presents significant robustness in most test-functions regardless of dimensionality while in most cases it was able to locate the global minima. Additionally, with respect to standard deviation, it is shown that for 5 out 6 objective functions the values are significantly low. It must be pointed out though, that in the case of test function 6, (Rosenbrock's function) PBA failed to converge to an acceptable average value of solutions. The result of the sensitivity analysis is a set of proposed value range per parameter that is presented in Table 2.6.

Investigating PBA performance

PBA performance is evaluated by comparing its results with those of up-to-date state-of-the-art metaheuristics known in literature [126, 28, 30]. The algorithms used in comparison are: nine versions of particle swam optimization algorithms [126], including the comprehensive learning particle swarm optimizer (CLPSO) algorithm proposed [126]: PSO with inertia weight (PSO-w) [189], PSO with constriction factor (PSO-cf) [189], local version of PSO with inertia weight (PSO-w-local) [104], local version of PSO with constriction factor (PSO-cflocal) [104], UPSO ([155], fully informed particle swarm (FIPS) [141], FDR-PSO [158], CPSO-H [210], and CLPSO [126]; five algorithms taken from the study by 28, including the IGSO proposed by [28]: SPSO, quantum behaved PSO (QPSO), weighted QPSO (WQPSO), group search optimizer (GSO) and improved GSO algorithm and nine particle swarm optimization algorithms taken for the study by [30], including the aging leader and challengers (ALC-PSO) algorithm proposed [30]: including the global version PSO (GPSO) with a fixed inertia weight $\omega = 0.4$ [189], the GPSO that decreases the value of ω linearly from 0.9 to 0.4 [190], the local version PSO with the RPSO [104], the one with the VPSO [104], FIPS [141], hierarchical PSO with HPSO-TVAC [171], DMS-PSO [129] and CLPSO [126].

For the first six test function, f_1 to f_6 in Table 2.3, PBA performance comparison is executed for dimensionality equal to 10-D and 30-D. In the case of the next seven test

Table 2.3 Test functions employed in the study

	Test Function	Domain	Optimum	Name
$f_1(x)$	$f(x) = \sum_{i=1}^{D} x_i^2$	$[-100,100]^D$	0	*Sphere*
$f_2(x)$	$f(x) = \sum_{i=1}^{D} [-x_i \sin(\sqrt{\lvert x_i \rvert})]$	$[-500,500]^D$	-418.9829^D	*Schwefel*
$f_3(x)$	$f(x) = \frac{1}{4000}(\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1)$	$[-600,600]^D$	0	*Griewank*
$f_4(x)$	$f(x) = 10D + \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i)]$	$[-100,100]^D$	0	*Rastrigin*
$f_5(x)$	$f(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (1-x_i)^2]$	$[-10,10]^D$	0	*Rosenbrock*
$f_6(x)$	$f(x) = -20\exp\left(-0.20\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + \exp(1)$	$[-32.768,32.768]^D$	0	*Ackley*
$f_7(x)$	$f(x) = \sum_{i=1}^{n}\left(\sum_{k=0}^{k_{\max}}\left[a^k\cos(2\pi b^k(x_i+0.5))\right]\right) - n\sum_{k=0}^{k_{\max}}[a^k\cos(2\pi b^k * 0.5)]$ $a = 0.5, b = 3, k_{\max} = 20$	$[-0.5,0.5]^D$	0	*Weierstrass*
$f_8(x)$	$f(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$	$[-100,100]^D$	0	*Quadric*
$f_9(x)$	$f(x) = \sum_{i=1}^{n}(x_i + 0.5)^2$	$[-100,100]^D$	0	*Step*
$f_{10}(x)$	$f(x) = -20\exp(-0.20\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi y_i)) + 20 + \exp(1)$, $y = M \cdot x$	$[-32.768,32.768]^D$	0	*Rotated Ackley*
$f_{11}(x)$	$f(x) = \frac{1}{4000}\sum_{i=1}^{n} y_i^2 - \prod_{i=1}^{n}\cos(\frac{y_i}{\sqrt{i}}) + 1$, $y = M \cdot x$	$[-600,600]^D$	0	*Rotated Griewank*
$f_{12}(x)$	$f(x) = 10n + \sum_{i=1}^{n}[y_i^2 - 10\cos(2\pi y_i)]$, $y = M \cdot x$	$[-5.12,5.12]^D$	0	*Rotated Rastrigin*
$f_{13}(x)$	$f(x) = 10n + \sum_{i=1}^{n}[z_i^2 - 10\cos(2\pi z_i)] - 330$, $z = (x-o) \cdot M$	$[-5.12,5.12]^D$	-330	*Shifted – Rotated Rastrigin*

Table 2.4 Statistical analysis for 10-D problems

Set	Sphere	Schwefel's	Griewank's	Rastrigin's	Rosenbrock's	Ackley's
1	1.05E-29	-4176.76	**0.00E+00**	**0.00E+00**	6.21E+00	7.99E-15
2	2.23E-72	-4171.97	1.01E-02	**0.00E+00**	6.12E+00	4.44E-15
3	3.38E-151	**-4188.91**	**0.00E+00**	**0.00E+00**	8.96E+00	**8.88E-16**
4	9.91E-24	-4178.68	1.06E-07	**0.00E+00**	7.17E+00	2.92E-13
5	8.11E-16	-4146.13	1.75E-01	**0.00E+00**	6.66E+00	6.67E-09
6	4.78E-70	-4168.50	**0.00E+00**	**0.00E+00**	8.96E+00	4.44E-15
7	1.31E-13	-4149.97	6.53E-12	2.04E-13	8.93E+00	1.76E-07
8	4.85E-24	-4184.25	1.34E-05	**0.00E+00**	**8.22E-02**	3.45E-12
9	5.68E-86	-4176.98	1.11E-01	**0.00E+00**	5.94E+00	7.99E-15
10	5.08E-08	-4187.64	2.28E-01	6.87E-08	8.03E+00	1.18E-04
11	7.89E-56	-4181.95	1.37E-01	**0.00E+00**	5.85E+00	4.44E-15
12	3.97E-39	-4169.06	1.04E-01	**0.00E+00**	5.87E+00	4.44E-15
13	1.61E-23	-4157.17	1.00E-12	**0.00E+00**	6.82E+00	1.67E-12
14	4.49E-20	-4164.68	2.85E-02	**0.00E+00**	8.72E+00	7.45E-11
15	4.36E-23	-4159.92	1.43E-01	**0.00E+00**	8.73E+00	3.61E-12
16	1.70E-32	-4177.25	**0.00E+00**	**0.00E+00**	7.13E+00	4.44E-15
17	8.75E-22	-4165.77	9.63E-02	**0.00E+00**	6.47E+00	1.58E-11
18	4.74E-08	-4184.97	1.38E-02	6.90E-08	7.39E+00	1.90E-04
19	6.23E-43	-4188.82	**0.00E+00**	**0.00E+00**	8.92E+00	4.44E-15
20	3.14E-14	-4172.84	6.27E-02	6.22E-14	5.47E+00	3.82E-06
21	**4.00E-188**	-4185.46	7.78E-02	**0.00E+00**	5.75E+00	4.44E-15
22	2.25E-17	-4149.44	2.90E-05	**0.00E+00**	8.74E+00	7.30E-10
23	1.47E-08	-4187.48	6.54E-03	1.04E-08	6.99E+00	3.99E-05
24	8.75E-13	-4181.78	2.37E-01	5.33E-15	7.17E+00	2.18E-08
25	3.38E-66	-4164.91	2.72E-01	**0.00E+00**	5.13E+00	2.93E-14
Average	4.52E-09	-4172.85	6.81E-02	5.92E-09	6.89E+00	1.41E-05
Standard Deviation	1.35E-08	12.72	8.48E-02	1.87E-08	1.85E+00	4.32E-05
Minimum	4.00E-188	-4188.91	0.00E+00	0.00E+00	8.22E-02	8.88E-16
Maximum	5.08E-08	-4146.13	2.72E-01	6.90E-08	8.96E+00	1.90E-04

Table 2.5 Statistical analysis for 30-D problems

Set	Sphere	Schwefel's	Griewank's	Rastrigin's	Rosenbrock's	Ackley's
1	1.19E-33	-12478.46	**0.00E+00**	**0.00E+00**	2.59E+01	7.99E-15
2	1.01E-83	-12400.32	1.70E-02	**0.00E+00**	2.88E+01	4.44E-15
3	4.98E-145	-12559.84	**0.00E+00**	**0.00E+00**	2.88E+01	**8.88E-16**
4	2.88E-25	-12460.17	4.02E-02	**0.00E+00**	2.86E+01	7.19E-14
5	4.36E-18	-12301.16	**0.00E+00**	**0.00E+00**	2.60E+01	6.56E-10
6	2.48E-77	-12414.98	1.93E-02	**0.00E+00**	2.59E+01	4.44E-15
7	3.84E-13	-12416.15	7.76E-13	1.51E-13	2.88E+01	1.25E-07
8	1.47E-25	-12533.16	**0.00E+00**	**0.00E+00**	2.71E+01	7.19E-14
9	1.59E-98	-12453.65	**0.00E+00**	**0.00E+00**	2.86E+01	2.58E-14
10	1.08E-07	-12558.81	6.70E-02	4.52E-08	2.80E+01	4.87E-05
11	6.01E-65	-12514.33	1.26E-02	**0.00E+00**	2.58E+01	7.99E-15
12	8.60E-45	-12294.99	1.09E-02	**0.00E+00**	**2.55E+01**	7.99E-15
13	5.11E-24	-12332.07	**0.00E+00**	**0.00E+00**	2.70E+01	3.42E-13
14	5.30E-21	-12425.66	**0.00E+00**	**0.00E+00**	2.88E+01	1.43E-11
15	4.90E-26	-12331.18	1.33E-02	**0.00E+00**	2.89E+01	5.77E-14
16	4.85E-36	-12437.18	**0.00E+00**	**0.00E+00**	2.79E+01	4.44E-15
17	9.55E-23	-12355.17	**0.00E+00**	**0.00E+00**	2.61E+01	1.24E-12
18	9.05E-07	-12540.62	3.40E-06	5.08E-07	2.69E+01	2.00E-04
19	9.31E-49	**-12561.68**	2.85E-02	**0.00E+00**	2.86E+01	4.44E-15
20	1.04E-15	-12353.39	1.01E-02	4.97E-14	2.56E+01	3.43E-07
21	**3.47E-206**	-12529.82	**0.00E+00**	**0.00E+00**	2.86E+01	3.44E+00
22	3.89E-18	-12353.25	**0.00E+00**	**0.00E+00**	2.80E+01	3.50E-10
23	5.69E-09	-12555.03	1.61E-07	1.60E-08	2.71E+01	2.30E-05
24	8.14E-16	-12530.70	1.33E-15	**0.00E+00**	2.70E+01	5.63E-09
25	1.89E-75	-12474.81	**0.00E+00**	**0.00E+00**	2.88E+01	3.58E+00
Average	4.07E-08	-12446.66	8.75E-03	2.27E-08	2.75E+01	2.81E-01
Standard Deviation	1.78E-07	86.79	1.58E-02	9.94E-08	1.22E+00	9.53E-01
Minimum	3.47E-206	-12561.68	0.00E+00	0.00E+00	2.55E+01	8.88E-16
Maximum	9.05E-07	-12294.99	6.70E-02	5.08E-07	2.89E+01	3.58E+00

Table 2.6 Proposed parameter values

Parameter	Description	Value Range
Npop	Population of pioneer patricles	[10, 50]
fnb	Neighbouring factor	[0.01, 0.10]
ftn	Fine tuning factor	[0.005, 0.05]
fms	Mid-scale factor	[0.40, 1.00]
fls	Large-scale factor	[1, 100]
pr	Probability for choosing large-scale search or memory consideration	[0, 1]
fFE	Function evaluations multiplication factor	[0.05, 0.25]

functions, f_7 to f_{13} in Table 2.3, comparison is performed on 30-D case. It must also be pointed out that initialization is random for all algorithms while the initialization range is equal to the described in Table 2.3 and not a decreased one as often used in literature. [126, 28, 30]. For comparison reasons, the termination criterion is the maximum function evaluations equal to 200,000 for all metaheuristic optimization algorithms [96].

Application of PBA to seven test functions

Wherever a rotation on a test function is mentioned, this points out that a linear, orthogonal transformation matrix M is applied such that F(M, s) is calculated, where the orthogonal (rotation) matrix M was generated using standard normally distributed entries by Gram-Schmidt orthonormalization [182]. The transformation matrix M is a pure rotation that applies no change to the structure of the function. All results for PBA applied to the test functions 7 to 13 for 30-D problems are presented in Table 2.7. Similarly to the sensitivity analysis, the average, standard deviation, minimum and maximum optimized objective function values achieved for 30 independent runs using the algorithmic parameters that presented the best performance for every test function examined are presented. These results are in accordance with the conclusions from the sensitivity analysis results, i.e. that PBA presents significant robustness and efficiency. In all of the seven test functions the best solution found is "equal" to the global minimum while even the worst optimization results is close to acceptable optimized. A convergence graph for test function 1 is presented in Figure 2.5 for the 30-D case while the variance of the optimized objective function value is also presented, where the fast convergence tendency of PBA is visible.

Table 2.7 PBA applied to test-functions 7 to 13 for 30-D problems

Function	**Mean Best**	**Minimum**	**Maximum**	**Deviation**
Weierstrass	7.11E-16	0.00E+00	1.42E-14	2.81E-15
Quadric	6.66E-23	1.36E-42	2.00E-21	3.58E-22
Step	3.77E-04	2.41E-04	4.95E-04	6.79E-05
Rotated Ackley's	4.44E-15	4.44E-15	4.44E-15	0.00E+00
Rotated Griewank's	8.59E-16	0.00E+00	2.55E-14	4.58E-15
Rotated Rastrigin's	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Shifted - Rotated Rastrigin's	-3.30E+02	-3.30E+02	-3.30E+02	0.00E+00

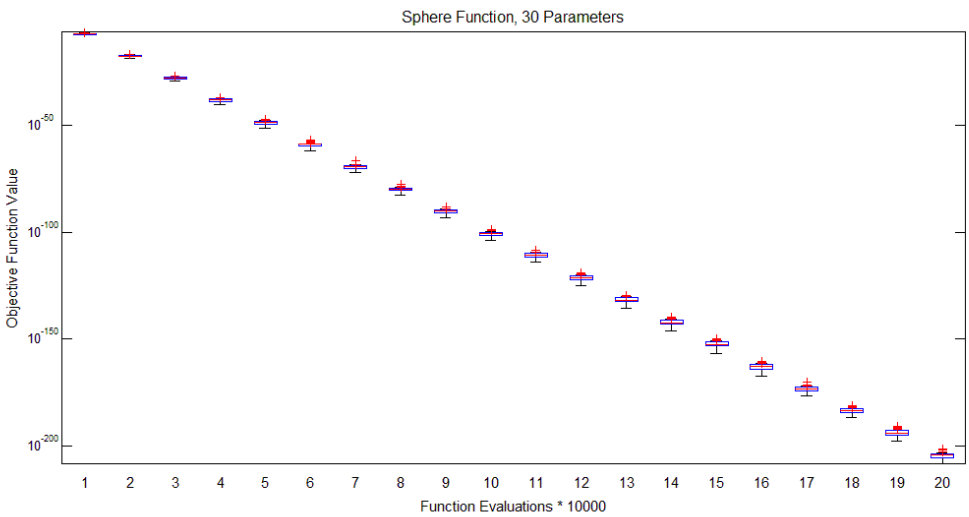


Fig. 2.5 Performance of the independent runs for different number of function evaluations for the test function (a) sphere for 30 parameters.

Comparison of PBA with state-of-the-art metaheuristics

For comparing PBA with the other metaheuristics, 10-D and 30-D problems are used, maximum objective function evaluations are set equal to 100,000 and 200,000 respectively while 30 independent runs are performed for each test function using the algorithmic parameters that had performed better in the previous test. The comparison is performed on the basis of the mean best objective function value calculated by each algorithm. The results are presented in Tables 2.8 and 2.9 for 10-D and 30-D problems. It can be seen that for the test functions used (i.e. f_1 to f_6) PBA managed to perform equally well or often even better than the other algorithms for both dimensions examined. In functions f_1 - f_4 and f_6 PBA achieved better mean best objective function values than the other algorithms and better standard deviation. Worth mentioning also that with respect to its performance for the Rosenbrock's function (f_6) performed just as well as the other algorithms both in terms of mean best objective function value and standard deviation.

PBA is also tested against the algorithms with the best performance in the CEC 2013 competition in the "Special Session and Competition on Real-Parameter Single Objective Optimization", iCMAES-ILS and NBIPOP-ACMA-ES [128, 132]. The comparison is done on the test functions 1, 2, 4, 10, 11 and 12 slightly modified than as presented in Table 2.3. A real number (F_k) was added in their expression, in particular $F_1 = -1400$ was added to test function 1, $F_2 = -100$, $F_4 = -400$, $F_{10} = -700$, $F_{11} = -500$ and $F_{12} = -300$ was added to the rest of the test functions respectively. The results of the comparison are presented in Table

2.10. The average error values correspond to the errors measured at the maximum number of function evaluations. Worth mentioning that for some of the test functions the design space used in CEC 2013 competition was wider while the function evaluations permitted for the 30-D problems employed in this comparative study were 300,000. In particular, in all these test functions (1, 2, 4, 10, 11 and 12) PBA achieved better performance than those obtained by the two algorithms.

Performance of PBA based on CEC 2014 benchmarks and complexity evaluation

PBA is also tested on the CEC 2014 test-suite for unconstrained optimization [127] on several problems with dimensionality equal to 10, 30, 50 and 100. In this suite, 30 test-functions are used where functions 1 to 3 are unimodal, functions 4 to 16 are multimodal, functions 17 to 22 are hybrid functions while 23 to 30 are composite ones. In these test functions the boundaries of the search space are $[-100, 100] * D$ while the maximum allowed function evaluations are equal to 10,000*D. Following the guidelines of the CEC 2014 competition, 51 independent runs were performed for each test function and the average error values were obtained. The results achieved for the CEC 2014 test-suite are provided in Tables 2.11, 2.12, 2.13 and 2.14 for the 10-D and 30-D, 50-D and 100-D problems, respectively. As seen in results, PBA manages to deal successfully with most of the test functions while it does not perform very well with some. By studying these four Tables, it is obvious that PBA is not to significantly affected by the dimension of the problem. In particular, with the increase of the dimensionality it is expected that the performance is decreased, however, this drop is relatively small to minimal. Some differences in results that are present between some test-functions used previously and in CEC 2014 test, it must be pointed out that they are the result of different search space used and different rotation methods defined in CEC 2014.

The Wilcoxon ranksum test [40] is also applied in order to present a more thorough performance evaluation for PBA against well-established algorithms that were also tested according to the standards of CEC 2014. In particular, simultaneous optimistic optimization (SOO) [167], fireworks algorithm with differential mutation (FWA-DM) [225], adaptive differential evolution (ADE), an improved variant of ADE with the use of partial opposition-based learning (POBL-ADE) [81] and the L-SHADE [208] algorithms where considered for this comparative study. The results of the Wilcoxon ranksum test are given in Table 2.15 where *Better* denotes the number of functions (out of the 30 ones) where a specific algorithm performs better compared to PBA, *Worst* denotes the number of functions where it performs worst and *Equal* denotes the number of functions where it has the same performance with

Table 2.8 Comparative study of PBA with state-of-the-art metaheuristics for test-functions 1 to 6 and 10-D problems

Algorithm	Sphere Function		Schwefel's Function		Griewank's Function		Rastrigin's Function		Rosenbrock's Function		Ackley's Function	
	Mean Best	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**
PSO-w	7.96E-51	NA	-3.87E+03	NA	9.69E-02	NA	5.82E+00	NA	3.08E+00	NA	1.58E-14	NA
PSO-cf	9.84E-105	NA	-3.20E+03	NA	1.19E-01	NA	1.25E+01	NA	**6.98E-01**	NA	9.18E-01	NA
PSO-w-local	2.13E-35	NA	-3.86E+03	NA	7.80E-02	NA	3.88E+00	NA	3.92E+00	NA	6.04E-15	NA
PSO-cflocal	1.37E-79	NA	-3.31E+03	NA	2.80E-02	NA	9.05E+00	NA	8.60E-01	NA	5.78E-02	NA
UPSO	9.84E-118	NA	-3.11E+03	NA	1.04E-01	NA	1.17E+01	NA	1.40E+00	NA	1.33E+00	NA
FDR-PSO	2.21E-90	NA	-3.34E+03	NA	9.24E-02	NA	7.51E+00	NA	8.67E+00	NA	3.18E-14	NA
FIPS	3.14E-30	NA	-4.12E+03	NA	1.31E-01	NA	2.12E+00	NA	2.78E+00	NA	3.75E-15	NA
CPSO-H	4.98E-40	NA	-3.98E+03	NA	4.07E-02	NA	**0.00E+00**	NA	1.53E+00	NA	1.49E-14	NA
CLPSO	5.15E-29	NA	**-4.19E+03**	NA	4.56E-03	NA	**0.00E+00**	NA	2.46E+00	NA	4.32E-14	NA
SPSO	3.27E-18	5.75E-18	-3.39E+03	9.06E+00	8.70E-02	5.70E-02	2.45E+00	1.63E+00	3.56E+01	5.69E+01	NA	NA
QPSO	7.40E-104	7.39E-103	-3.48E+03	6.04E+00	3.58E-04	2.80E-02	2.31E+00	2.00E-02	7.43E+00	3.20E-01	NA	NA
WQPSO	8.37E-106	1.08E-106	-3.77E+03	4.33E+00	1.63E-04	1.63E-04	1.84E+00	1.00E-02	1.04E+01	**2.50E-01**	NA	NA
GSO	6.75E-18	1.88E-18	-4.03E+03	7.10E+00	1.60E-01	4.40E-02	2.57E+00	1.82E+00	4.38E+00	1.97E+00	NA	NA
IGSO	7.33E-41	1.89E-41	**-4.19E+03**	3.10E+00	4.38E-05	1.40E-02	6.70E-01	4.20E-01	7.60E-01	5.60E-01	NA	NA
PBA	**5.37E-187**	**0.00E+00**	**-4.19E+03**	**1.02E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	5.81E+00	1.00E+00	**8.88E-16**	**0.00E+00**

Table 2.9 Comparative study of PBA with state-of-the-art metaheuristics for test-functions 1 to 6 and 30-D problems

Algorithm	Sphere Function		Schwefel's Function		Griewank's Function		Rastrigin's Function		Rosenbrock's Function		Ackley's Function	
	Mean Best	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**	**Mean Best**	**St. Var.**
ALC-PSO	1.68E-161	8.21E-161	-1.25E+04	5.41E+01	1.22E-02	1.58E-02	2.53E-14	1.38E-14	7.61E+00	6.66E+00	1.15E-14	2.94E-15
GPSO	2.65E-161	2.38E-161	-8.86E+03	5.20E+02	1.51E-02	1.75E-02	5.76E+01	1.46E+01	1.17E+01	1.50E+01	1.60E+00	1.03E+00
GPSO-2	7.08E-53	1.71E-52	-1.12E+04	3.33E+02	1.65E-02	1.69E-02	2.52E+01	5.21E+00	2.55E+01	2.59E+01	1.10E-14	2.27E-15
VPSO	1.90E-38	3.99E-38	-9.88E+03	5.20E+02	2.41E-02	2.25E-02	2.92E+01	9.66E+00	2.95E+01	2.47E+01	1.52E-14	4.10E-15
RPSO	5.58E-29	1.42E-28	-9.68E+03	3.44E+02	8.17E-03	1.78E-02	3.88E+01	8.63E+00	2.06E+01	1.25E+01	2.66E-14	5.45E-14
CLPSO	1.39E-27	2.05E-27	**-1.26E+04**	3.61E+01	2.01E-14	8.67E-14	2.44E-14	5.98E-14	1.70E+01	1.28E+01	2.49E-14	4.18E-15
HPSO-TVAC	1.45E-41	4.64E-41	-1.10E+04	2.61E+02	1.39E-02	1.39E-02	1.43E+00	2.96E+00	1.20E+01	1.61E+01	8.95E-11	4.46E-10
FIPS	2.60E-30	3.24E-30	-1.05E+04	3.87E+02	2.47E-04	2.47E-04	2.87E+01	1.46E+01	2.25E+01	4.39E-01	7.58E-15	6.49E-16
DMS-PSO	1.95E-54	8.43E-54	-9.63E+03	4.78E+02	1.07E-02	1.60E-02	2.78E+01	7.57E+00	1.95E+01	1.20E+01	9.23E-15	1.79E-15
PSO-w	9.78E-30	NA	-1.15E+04	NA	8.13E-03	NA	2.90E+01	NA	2.93E+01	NA	3.94E-14	NA
PSO-cf	5.88E-100	NA	-8.79E+03	NA	2.06E-02	NA	5.62E+01	NA	1.11E+01	NA	1.12E+00	NA
PSO-w-local	5.35E-100	NA	-1.10E+04	NA	5.91E-03	NA	2.72E+01	NA	2.39E+01	NA	9.10E-08	NA
PSO-cflocal	7.70E-54	NA	-8.79E+03	NA	5.91E-03	NA	4.53E+01	NA	1.71E+01	NA	5.33E-15	NA
UPSO	4.17E-87	NA	-7.73E+03	NA	1.66E-03	NA	6.59E+01	NA	1.51E+01	NA	**1.22E-15**	NA
FDR-PSO	4.88E-102	NA	-8.96E+03	NA	1.01E-02	NA	2.84E+01	NA	**5.39E+00**	NA	2.84E-14	NA
FIPS	2.69E-12	NA	-1.05E+04	NA	1.16E-06	NA	7.30E+01	NA	2.45E+01	NA	4.81E-07	NA
CPSO-H	1.16E-113	NA	-1.15E+04	NA	3.63E-02	NA	**0.00E+00**	NA	7.08E+00	NA	4.93E-14	NA
CLPSO	4.46E-14	NA	**-1.26E+04**	NA	3.14E-10	NA	4.85E-10	NA	2.10E+01	NA	0.00E+00	NA
SPSO	3.26E-12	5.33E-12	-9.79E+03	4.20E+01	1.60E-02	9.00E-03	2.92E+01	1.11E+01	2.13E+02	2.80E+02	NA	NA
QPSO	1.68E-51	6.70E-52	-1.02E+04	3.11E+01	5.47E-05	1.40E-02	1.60E+01	2.00E-02	5.35E+01	1.87E+00	NA	NA
WQPSO	3.56E-60	2.74E-62	-1.12E+04	2.82E+01	3.03E-05	1.58E-05	1.51E+01	4.00E-02	5.11E+01	**3.50E-01**	NA	NA
GSO	8.78E-12	2.13E-12	-1.23E+04	2.17E+01	3.30E-02	3.00E-03	1.46E+01	4.38E+00	4.07E+01	2.09E+01	NA	NA
IGSO	2.16E-21	1.22E-21	-1.25E+04	**9.17E+00**	4.09E-05	1.40E-02	8.79E+00	2.33E+00	1.05E+01	1.37E+00	NA	NA
PBA	**5.32E-203**	**0.00E+00**	**-1.26E+04**	1.87E+01	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	2.48E+01	4.93E-01	3.73E-15	**1.45E-15**

Table 2.10 Comparative study of PBA with state-of-the-art metaheuristics for 30-D problems

Algorithm	iCMAES-ILS	NBIACMA	PBA
Sphere Function	1.00E-08	1.00E-08	**5.32E-203**
Schwefel's Function	7.08E+02	8.10E+02	**5.22E+00**
Rastrigin's Function	2.25E+00	3.04E+00	**0.00E+00**
Rotated Ackley's Function	2.09E+01	2.09E+01	**4.44E-15**
Rotated Griewank's Function	1.00E-08	1.00E-08	**8.59E-16**
Rotated Rastrigin's Function	1.72E+00	2.91E+00	**0.00E+00**

PBA. It should be underlined that PBA appears to be less affected by the dimension of the problem compared to most of the other algorithms.

The complexity of PBA is also calculated according to the CEC 2014 benchmark suite guide[127] and is presented in Table 2.16 for all dimensionality cases of the CEC 2014. As suggested, T_0 is the time required to perform the calculations described in Eq. 2.36,

$$\begin{aligned}
&for\ i = 1 : 10^6 \\
&x = 0.55 + (double)i;\ x = x + x;\ x = x/2; \\
&x = x * x;\ x = sqrt(x);\ x = \log(x); \\
&x = \exp(x);\ x = x/(x+2); \\
&end
\end{aligned} \tag{2.36}$$

T_1 is the time needed for executing 200,000 evaluations of test-function No.18 and T_2 is the time the algorithmic procedure needs to perform 200,000 function evaluations of No.18. T_1 and T_2 are scaled linearly with dimensions as by the linear growth of $(T_2 - T_1)/T_0$. It must also be noted that all test runs were performed on a computer with the following specifications: Windows 7 OS, Intel i7 3610QM CPU, 12GB RAM using Matlab programming language. The complexity of PBA is presented in Table 2.16. As it can be seen, computational load of PBA is quite small. It must be pointed out that the time needed for executing 200,000 function evaluations of a 100-D problem is less than 10 seconds.

Advantages of PBA

By following the main practices addressed in modern literature, the advantages of PBA were tested on three different terms:

- The sensitivity of the performance was assessed with reference to its algorithmic parameters.

Table 2.11 Performance of PBA using the CEC 2014 test suite (10-D)

Func.	Best	Worst	Median	Mean	Std.
F1	1.13E+05	9.16E+06	6.20E+05	1.66E+06	2.06E+067
F2	4.53E+05	5.12E+07	9.27E+06	1.09E+07	8.64E+06
F3	4.11E+02	4.34E+03	1.19E+03	1.26E+03	6.80E+02
F4	4.02E+02	4.43E+02	4.14E+02	4.20E+02	1.49E+01
F5	5.05E+02	5.20E+02	5.20E+02	5.18E+02	4.57E+00
F6	6.02E+02	6.06E+02	6.03E+02	6.03E+02	9.72E-01
F7	7.01E+02	7.02E+02	7.01E+02	7.01E+02	1.68E-01
F8	8.03E+02	8.15E+02	8.07E+02	8.08E+02	2.73E+00
F9	9.06E+02	9.21E+02	9.14E+02	9.15E+02	3.47E+00
F10	1.03E+03	1.39E+03	1.07E+03	1.09E+03	6.25E+01
F11	1.23E+03	1.88E+03	1.48E+03	1.48E+03	1.33E+02
F12	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.14E-01
F13	1.30E+03	1.30E+03	1.30E+03	1.30E+03	6.64E-02
F14	1.40E+03	1.40E+03	1.40E+03	1.40E+03	7.34E-02
F15	1.50E+03	1.51E+03	1.50E+03	1.50E+03	8.25E-01
F16	1.60E+03	1.60E+03	1.60E+03	1.60E+03	4.11E-01
F17	2.14E+03	6.13E+05	5.09E+04	1.76E+05	2.08E+05
F18	1.88E+03	2.01E+04	4.46E+03	7.27E+03	5.26E+03
F19	1.90E+03	1.90E+03	1.90E+03	1.90E+03	3.72E-01
F20	2.02E+03	1.14E+04	2.30E+03	3.08E+03	2.18E+03
F21	2.39E+03	3.11E+04	6.33E+03	7.74E+03	5.41E+03
F22	2.21E+03	2.24E+03	2.23E+03	2.23E+03	4.92E+00
F23	2.50E+03	2.63E+03	2.51E+03	2.54E+03	5.44E+01
F24	2.51E+03	2.54E+03	2.52E+03	2.52E+03	5.68E+00
F25	2.61E+03	2.70E+03	2.65E+03	2.66E+03	2.84E+01
F26	2.70E+03	2.70E+03	2.70E+03	2.70E+03	6.96E-02
F27	2.70E+03	3.10E+03	2.71E+03	2.87E+03	1.88E+02
F28	3.18E+03	3.32E+03	3.23E+03	3.23E+03	2.93E+01
F29	3.20E+03	3.79E+03	3.36E+03	3.37E+03	1.23E+02
F30	3.62E+03	4.87E+03	4.04E+03	4.04E+03	2.43E+02

Table 2.12 Performance of PBA using the CEC 2014 test suite (30-D)

Func.	Best	Worst	Median	Mean	Std.
F1	7.09E+06	1.00E+08	2.94E+07	3.50E+07	2.16E+07
F2	4.83E+07	9.61E+08	2.60E+08	3.05E+08	1.89E+08
F3	9.05E+02	2.13E+04	7.07E+03	6.97E+03	3.96E+03
F4	4.97E+02	6.53E+02	5.84E+02	5.78E+02	3.62E+01
F5	5.20E+02	5.21E+02	5.21E+02	5.21E+02	5.26E-02
F6	6.11E+02	6.21E+02	6.16E+02	6.16E+02	2.40E+00
F7	7.01E+02	7.08E+02	7.04E+02	7.04E+02	1.72E+00
F8	8.29E+02	8.94E+02	8.53E+02	8.56E+02	1.48E+01
F9	9.77E+02	1.04E+03	1.01E+03	1.01E+03	1.33E+01
F10	1.30E+03	2.87E+03	1.84E+03	1.89E+03	3.68E+02
F11	3.36E+03	5.45E+03	4.54E+03	4.49E+03	5.35E+02
F12	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.43E-01
F13	1.30E+03	1.30E+03	1.30E+03	1.30E+03	9.49E-02
F14	1.40E+03	1.40E+03	1.40E+03	1.40E+03	4.57E-02
F15	1.51E+03	1.53E+03	1.52E+03	1.52E+03	3.36E+00
F16	1.61E+03	1.61E+03	1.61E+03	1.61E+03	3.78E-01
F17	4.94E+05	8.42E+06	2.76E+06	3.40E+06	2.12E+06
F18	1.31E+05	5.23E+06	1.60E+06	1.70E+06	1.06E+06
F19	1.91E+03	1.94E+03	1.91E+03	1.91E+03	5.23E+00
F20	3.26E+03	2.04E+04	7.12E+03	8.77E+03	4.31E+03
F21	4.11E+04	2.09E+06	3.35E+05	4.39E+05	3.40E+05
F22	2.29E+03	2.82E+03	2.53E+03	2.53E+03	1.08E+02
F23	2.50E+03	2.62E+03	2.62E+03	2.60E+03	3.88E+01
F24	2.60E+03	2.62E+03	2.61E+03	2.61E+03	3.98E+00
F25	2.70E+03	2.71E+03	2.70E+03	2.70E+03	1.41E+00
F26	2.70E+03	2.80E+03	2.70E+03	2.70E+03	1.93E+01
F27	3.11E+03	3.42E+03	3.12E+03	3.14E+03	6.53E+01
F28	3.00E+03	4.20E+03	3.94E+03	3.90E+03	2.16E+02
F29	1.16E+04	1.09E+05	2.99E+04	3.59E+04	2.18E+04
F30	7.26E+03	2.93E+04	1.45E+04	1.55E+04	4.24E+03

Table 2.13 Performance of PBA using the CEC 2014 test suite (50-D)

Func.	Best	Worst	Median	Mean	Std.
F1	1.37E+07	1.33E+08	3.28E+07	3.67E+07	1.87E+07
F2	1.24E+08	2.04E+09	7.75E+08	8.22E+08	4.71E+08
F3	5.39E+03	3.17E+04	1.54E+04	1.59E+04	4.72E+03
F4	5.86E+02	8.44E+02	7.04E+02	7.03E+02	5.50E+01
F5	5.21E+02	5.21E+02	5.21E+02	5.21E+02	4.56E-02
F6	6.23E+02	6.40E+02	6.32E+02	6.32E+02	3.45E+00
F7	7.03E+02	7.20E+02	7.08E+02	7.09E+02	4.57E+00
F8	8.60E+02	1.00E+03	9.18E+02	9.20E+02	3.28E+01
F9	1.07E+03	1.15E+03	1.13E+03	1.13E+03	1.83E+01
F10	1.88E+03	4.98E+03	3.34E+03	3.36E+03	7.68E+02
F11	6.76E+03	9.65E+03	8.60E+03	8.43E+03	6.21E+02
F12	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.67E-01
F13	1.30E+03	1.30E+03	1.30E+03	1.30E+03	8.29E-02
F14	1.40E+03	1.40E+03	1.40E+03	1.40E+03	4.84E-02
F15	1.53E+03	1.57E+03	1.54E+03	1.54E+03	8.71E+00
F16	1.62E+03	1.62E+03	1.62E+03	1.62E+03	4.77E-01
F17	2.00E+06	2.43E+07	7.27E+06	7.72E+06	3.93E+06
F18	2.38E+06	4.58E+07	1.05E+07	1.25E+07	7.99E+06
F19	1.92E+03	2.04E+03	1.95E+03	1.95E+03	2.49E+01
F20	8.94E+03	3.13E+04	1.85E+04	1.76E+04	4.26E+03
F21	5.41E+05	5.81E+06	2.45E+06	2.60E+06	1.34E+06
F22	2.56E+03	3.57E+03	3.18E+03	3.14E+03	2.14E+02
F23	2.51E+03	2.68E+03	2.66E+03	2.64E+03	4.20E+01
F24	2.60E+03	2.67E+03	2.62E+03	2.62E+03	1.06E+01
F25	2.70E+03	2.73E+03	2.70E+03	2.70E+03	3.60E+00
F26	2.70E+03	2.80E+03	2.70E+03	2.70E+03	1.38E+01
F27	3.18E+03	4.05E+03	3.87E+03	3.83E+03	1.49E+02
F28	3.19E+03	5.59E+03	4.86E+03	4.81E+03	3.54E+02
F29	4.23E+04	8.02E+05	2.23E+05	2.87E+05	1.81E+05
F30	2.16E+04	1.11E+05	3.85E+04	4.16E+04	1.53E+04

Table 2.14 Performance of PBA using the CEC 2014 test suite (100-D)

Func.	Best	Worst	Median	Mean	Std.
F1	7.61E+07	3.03E+08	1.36E+08	1.47E+08	4.65E+07
F2	4.09E+08	6.37E+09	2.04E+09	2.31E+09	1.38E+09
F3	1.46E+04	4.94E+04	2.79E+04	2.96E+04	6.94E+03
F4	8.57E+02	1.38E+03	1.07E+03	1.06E+03	1.30E+02
F5	5.21E+02	5.21E+02	5.21E+02	5.21E+02	3.73E-02
F6	6.55E+02	6.92E+02	6.81E+02	6.78E+02	9.55E+00
F7	7.05E+02	7.66E+02	7.22E+02	7.24E+02	1.45E+01
F8	9.78E+02	1.27E+03	1.12E+03	1.11E+03	6.99E+01
F9	1.34E+03	1.61E+03	1.52E+03	1.51E+03	4.89E+01
F10	4.09E+03	1.22E+04	8.09E+03	7.87E+03	2.00E+03
F11	1.63E+04	2.15E+04	1.94E+04	1.94E+04	1.17E+03
F12	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.61E-01
F13	1.30E+03	1.30E+03	1.30E+03	1.30E+03	6.15E-02
F14	1.40E+03	1.40E+03	1.40E+03	1.40E+03	4.82E-02
F15	1.57E+03	1.73E+03	1.59E+03	1.61E+03	3.52E+01
F16	1.64E+03	1.64E+03	1.64E+03	1.64E+03	6.17E-01
F17	1.08E+07	6.82E+07	3.21E+07	3.34E+07	1.26E+07
F18	1.31E+07	1.74E+08	6.45E+07	6.78E+07	3.76E+07
F19	1.96E+03	2.12E+03	2.07E+03	2.07E+03	2.86E+01
F20	3.34E+04	9.53E+04	7.83E+04	7.30E+04	1.69E+04
F21	3.56E+06	3.59E+07	1.43E+07	1.54E+07	6.75E+06
F22	3.88E+03	5.53E+03	4.93E+03	4.87E+03	3.49E+02
F23	2.50E+03	2.75E+03	2.67E+03	2.66E+03	6.40E+01
F24	2.60E+03	2.77E+03	2.65E+03	2.66E+03	2.75E+01
F25	2.70E+03	2.75E+03	2.71E+03	2.71E+03	6.88E+00
F26	2.70E+03	2.80E+03	2.80E+03	2.77E+03	4.65E+01
F27	4.03E+03	5.35E+03	4.87E+03	4.81E+03	2.94E+02
F28	3.45E+03	9.91E+03	7.78E+03	7.70E+03	1.40E+03
F29	1.96E+05	3.63E+06	1.09E+06	1.51E+06	1.06E+06
F30	4.00E+04	7.59E+05	2.58E+05	2.64E+05	1.36E+05

Table 2.15 PBA compared to other algorithms – Wilcoxon's ranksum test

	With reference to PBA	**10-D**	**30-D**	**50-D**	**100-D**
SOO [49]	Better	12	12	14	13
	Worst	17	15	15	16
	Equal	1	3	1	1
FWA-DM [50]	Better	26	22	21	14
	Worst	3	5	7	15
	Equal	1	3	2	1
ADE [51]	Better	-	-	16	22
	Worst	-	-	14	8
	Equal	-	-	0	0
ADE-POBL [51]	Better	27	19	20	22
	Worst	2	8	9	7
	Equal	1	3	1	1
L-SHADE [52]	Better	27	25	24	24
	Worst	2	3	5	5
	Equal	1	2	1	1

Table 2.16 PBA complexity in seconds

	To	**T1**	**T2**	**(T2-T1)/To**
D=10	0.14	1.53	2.52	7.16
D=30	0.14	1.84	3.77	14.00
D=50	0.14	2.2	5.05	20.62
D=100	0.14	3.77	9.27	39.91

- The algorithm was assessed in comparison with other state-of-the-art algorithms and in particular against the best algorithms that were identified during the CEC 2013 competition presented in the "Special Session and Competition on Real-Parameter Single Objective Optimization" [128].
- The performance of the algorithm was assessed based on the CEC 2014 benchmark test suite [127].

In general it can be said that the main strength of PBA is its robustness and efficiency. In particular, as it was identified through these tests the algorithms is not influenced by the algorithmic parameters, the computational effort required internally is low, it is not influenced significantly by the increase of the dimensionality while in most of the test functions outperforms many state-of-the-art metaheuristics. While many variants of existing evolutionary and swarm algorithms, to the authors' knowledge there is no similarity with any other existing algorithm. More specifically, the rules based on which new position vectors are generated are completely different to existing evolutionary and swarm optimization algorithms. For example, in PBA there are no crossover/recombination or mutation operators; while there is also no velocity vector neither exchange of experience using the cognitive and social parameters. The similarity to PSO and variants is limited to the terms used for the position vectors (particles).

2.4.2 Optimal structures inspection following a seismic event

Catastrophic events like earthquakes could seriously affect structural and operational condition of civil infrastructures, leading to severe economic losses for the local or national or even global economy. It is of great importance to mitigate such impacts and risks through careful planning. Disaster management is a multi-stage process starting with the pre-disaster planning and system improvement, and extending to post-disaster system response, recovery and reconstruction. Pre-disaster planning stage involves strategic decision-making for risk assessment and management, infrastructure improvements to reduce vulnerability, enhanced human and physical system resilience, and emergency plans. Post-disaster stage involves tactical and operational decision-making for providing critical emergency, recovery and re-construction services [10, 26], to support society. Following such disasters, local communities and search-rescue crews are faced with rapidly degrading infrastructure networks that may result in much slower response times, delays in population evacuation, and significant complications in infrastructure repair. Recent advances on computational engineering and optimization have enabled the transition from traditional trial-and-error procedures to fully

automated ones, where search algorithms are used. This is mostly attributed to the rapid development of metaheuristic search algorithms. Minimizing inspection times in post-disaster management requires optimal scheduling of the inspection crews which is a complex combinatorial problem. In this thesis an improved harmony search algorithm (IHS) is proposed for solving the districting problem where an urban area is decomposed into optimal areas equal to the number of available inspection crews. IHS algorithm is applied in two cities:

- the city of Patras and
- the city of Thessaloniki,

both located in Greece. The proposed improved algorithm is compared with three well established nature inspired algorithms and the random search (RS) procedure when implemented for solving the districting problem. In particular, the basic formulation of the harmony search (HS) [60], the particle swarm optimization (PSO) [103] and the differential evolution (DE) [201] algorithms are employed.

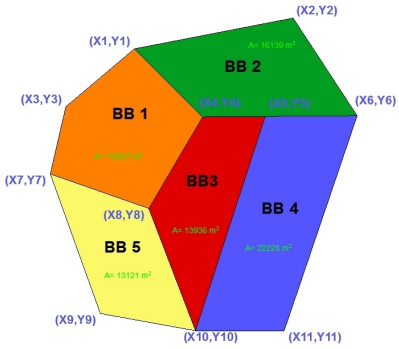
Defining optimal districting problem

Urban areas are consist of building) blocks (SB_k) that can be defined by their nodal coordinates (X_k, Y_k) while there are specific terms related to the construction for each SB_k that have to be respected. Such terms are: the use of land (u_L), the building factor (f_B), the maximum built area ($A_{B,max}$) and the coverage factor (f_C). The use of land determines the usage of the constructions allowed at a specific structural block (houses, factories, etc.). The building factor is the ratio between the maximum permitted area of constructions for the specific SB_k, divided by the area of the structural block, while the maximum permitted area of constructions for SB_k defines $A_{B,max}$. For example, if the structural block area is equal to $1000m^2$ and the corresponding SB_k is equal to 1.6, then $A_{B,max}$ is equal to $10001.6 = 1600m^2$. Fig. 2.6 depicts an example of neighboring structural blocks with various areas, building factors and thus different maximum built areas A_B, max. The districting problem aims to partition the urban area into groups of structural blocks, called districts, having almost the same sum of the corresponding maximum built areas $A_{B,max}$. As it will be presented in the next section, the inspection demand D_i is used in the formulation that is defined according to expression:

$$D_i = \sum_{k=1}^{n_{SB}^{(i)}} D(k), i = 1, \ldots N_{IC} \text{ and } D(k) = A(k) \cdot f_B(k) \tag{2.37}$$

where n_{SB}^i is the number of structural blocks assigned to the $i_t h$ district while the structural block total area and the corresponding building factor for the $k_t h$ structural block is equal

to $A(k)$ and $f_B(k)$. For the current problem, it is assumed that the allowed building factor is covered as it is the typical situation in Greece.



	BB 1	BB 2	BB3	BB 4	BB 5
AREA A (m²)	16557	16139	13936	22228	13121
MB	1.60	1.80	2.30	1.10	2.00
MBA (m²)	26491.20	29050.20	32052.80	24450.80	26242.00

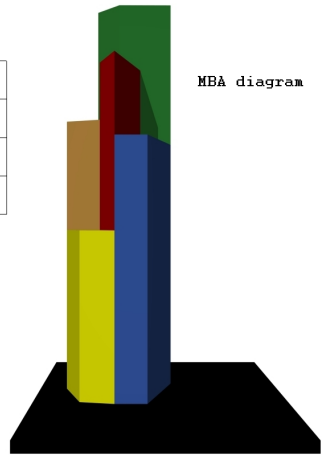


Fig. 2.6 Building blocks, Area, f_B, $A_{B,max}$

The main objective of the districting problem in post-disaster management is to define areas of responsibility for available inspection crews; the problem is formulated as a nonlinear programming optimization problem as follows [91]:

$$\min \sum_{i=1}^{N_{IC}} \sum_{k=1}^{n_{SB}^{(i)}} \left[\frac{D(k)}{U_{in}} + \frac{d(SB_k, C_i)}{U_{tr}} \right] \cdot \sigma_{IC} \tag{2.38}$$

where N_{IC} is the number of the available inspection crews, $n_{SB}^{(}i)$ is the number of building blocks assigned to the i_{th} district-inspection crew, $d(SB_k, C_i)$ is the distance between the SB_k building block and C_i is the starting block of the crew responsible for the i_{th} group of structural blocks, U_{in} is the inspection speed of the inspection crews, and U_{tr} is the traveling speed of the inspection crews. D_k is inspection "demand" for the k_{th} building block defined as the product of the building block total area $A(k)$ times the building factor $f_B(k)$ (i.e. the structured percentage of the area) and σ_{IC} is the value of the standard deviation of working hours of all inspection crews. Thus, the districting problem is formulated as a discrete unconstrained nonlinear optimization problem where the objective is to define which inspection crew is responsible for which block. Therefore, the design variables are integers denoting the inspection crew to which each built-up block has been assigned to.

In order to assess the performance of HS, IHS, PSO, DE algorithms and the random search (RS) procedure, they are implemented into post-disaster management problems. In particular, two test cases are considered based on an imaginary seismic event for the cities of Patras and Thessaloniki in Greece. Following this imaginary seismic event, all structures

need to be inspected for restoration of the urban activities. Therefore, the first step of the post-disaster management procedure is to optimally assign the structural blocks into the available inspection crews (districting problem).

Patras is a medium-sized city while Thessaloniki is the second largest city of Greece. The city of Patras is composed by 112 grouped building blocks while the total built area is equal to 18559676 m^2. On the other hand, the city of Thessaloniki is composed of 471 grouped building blocks and its built area is equal to 154205128 m^2. The difference in size of the two cities will offer information on the performance of the proposed IHS algorithm in medium and larger scale problems. For the city of Patras, two problems are examined varying on the number of available inspection crews. By increasing the number of available inspection crews, the complexity of the problem is also increased offering us useful information regarding the performance of the algorithms. In such a problem the computational cost is proportional to the number of the building blocks; therefore, by grouping the building blocks into larger ones, significant reduction on the computational demand is achieved. The grouping is applied into neighboring building blocks with similar build up percentages. For an unbiased comparison of the algorithms the termination criterion for all implementations examined in this part of the thesis is the same (200,000 function evaluations).

Post-Disaster management in the city of Patras

The city of Patras is composed by $N_{SB} = 112$ grouped building blocks with various areas and built-up percentages (see Figure 2.7). Two different problems (A and B) are examined with the hypothesis that ten and thirty crews are available for performing the inspection. By separating them into two 8-hour shift work-groups, $N_{IC} = 5$, inspection crews are available for 16 inspection hours per day (denoted as problem A) and $N_{IC} = 15$, inspection crews are available for 16 inspection hours per day (denoted as problem B).

In the first part of the study, sensitivity analysis is performed on the problem described in Eq. 2.38 in order to examine the influence of parameter values on the robustness of the IHS algorithm in solving the districting problem. In order to increase the quality of the sensitivity analysis, a sampling method named Latin hypercube sampling (LHS) is used [152] to generate combinations of the parameters of IHS. LHS ensures that all regions of the sample space of the parameters will be sampled. For each implementation, 32 independent runs are carried out, corresponding to the different sets of the parameters. The parameters sampled with LHS for IHS are HMS and HMCR, Table 2.17 depicts the combinations of the parameters. HMS defines the size of the memory used by HS for storing solution vectors defined in the range of [5, 20] while HMCR ($0 \leq HMCR \leq 1$) is the probability for random selection [94].

Table 2.17 Parameters combinations for sensitivity analysis of IHS, PSO and DE algorithms

IHS		**PSO**				**DE**			
HMS	HMCR	NP	c1	c2	w	NP	λ	CR	F
11	0.09	158	0.648	0.387	0.474	106	0.648	0.387	0.942
20	0.139	109	0.108	0.112	0.362	113	0.108	0.112	0.779
11	0.002	142	0.774	0.184	0.105	139	0.774	0.184	0.187
12	0.098	61	0.408	0.431	0.157	151	0.408	0.431	0.194
15	0.269	119	0.364	0.524	0.541	185	0.364	0.524	0.896
17	0.24	102	0.132	0.716	0.137	200	0.132	0.716	0.653
8	0.049	121	0.97	0.829	0.391	156	0.97	0.829	0.079
6	0.231	150	0.4	0.147	0.491	172	0.4	0.147	0.524
20	0.072	128	0.796	0.976	0.054	56	0.796	0.976	0.71
18	0.186	174	0.314	0.883	0.278	96	0.314	0.883	0.921
9	0.146	179	0.049	0.259	0.664	85	0.049	0.259	0.022
10	0.173	188	0.731	0.075	0.239	176	0.731	0.075	0.118
7	0.25	155	0.281	0.028	0.198	73	0.281	0.028	0.232
16	0.016	96	0.011	0.306	0.209	125	0.011	0.306	0.283
7	0.26	85	0.817	0.595	0.033	148	0.817	0.595	0.404
8	0.208	64	0.562	0.729	0.638	189	0.562	0.729	0.971
15	0.108	70	0.492	0.226	0.42	192	0.492	0.226	0.33
5	0.059	59	0.092	0.451	0.333	64	0.092	0.451	0.555
5	0.044	164	0.459	0.053	0.628	161	0.459	0.053	0.144
16	0.198	133	0.163	0.645	0.698	77	0.163	0.645	0.73
10	0.075	171	0.617	0.313	0.576	118	0.617	0.313	0.565
13	0.165	193	0.907	0.193	0.568	79	0.907	0.193	0.47
12	0.151	197	0.584	0.358	0.446	137	0.584	0.358	0.858
17	0.189	148	0.954	0.562	0.256	178	0.954	0.562	0.371
14	0.289	113	0.247	0.852	0.014	132	0.247	0.852	0.274
9	0.122	82	0.204	0.751	0.414	166	0.204	0.751	0.809
19	0.221	186	0.525	0.475	0.079	68	0.525	0.475	0.824
13	0.128	135	0.299	0.578	0.167	98	0.299	0.578	0.618
6	0.293	99	0.668	0.668	0.339	104	0.668	0.668	0.667
18	0.027	77	0.707	0.967	0.3	123	0.707	0.967	0.428
19	0.272	90	0.881	0.918	0.611	91	0.881	0.918	0.459
14	0.036	52	0.863	0.798	0.514	50	0.863	0.798	0.057

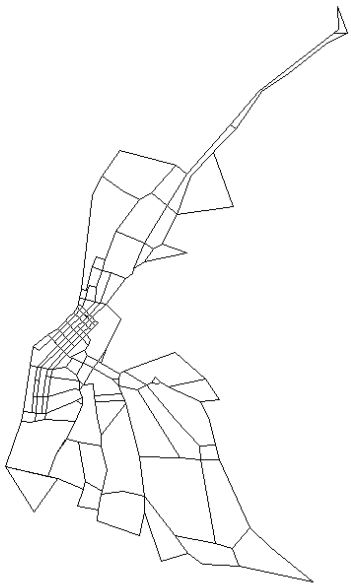


Fig. 2.7 City of Patras - Subdivision into building blocks

Since the inspection time varies between the crews, the maximum time required between the inspection crews is considered as the time required for the inspection procedure of the entire city. Figure 2.8 depicts the time required for the inspection of the building blocks of the city of Patras for each combination of parameters. These results are used for evaluating the performance of IHS with respect to parameter values. An indicator of the quality of the results is the difference between the working hours of the different inspection crews. As can be seen in Figure 2.8, IHS is not sensitive to the parameters of the algorithm since it converges to the same optimal solution in almost all the combinations of parameters. Additionally, Figure 2.10 presents the variance of the inspection time between the crews for the combination of parameters. IHS algorithm presents small variance values regardless the parameters used. Figure 2.10 depicts the total number of iterations required for converging to the optimal solution.

The inspection time of the best solutions found from 1288.55 to 1310.44 hours; the mean time varies from 1238.17 to 1238.22 hours while the inspection time for all crews varies from 1145.72 to 1310.44 hours. In particular, Figure 2.11 presents the optimal districting solution that obtained by IHS for the formulation of Eq. 2.38. The above mentioned results indicate that IHS is capable of solving the districting problem of Eq. 2.38. In the second part of the study performed for the city of Patras, four different nature inspired algorithms and a

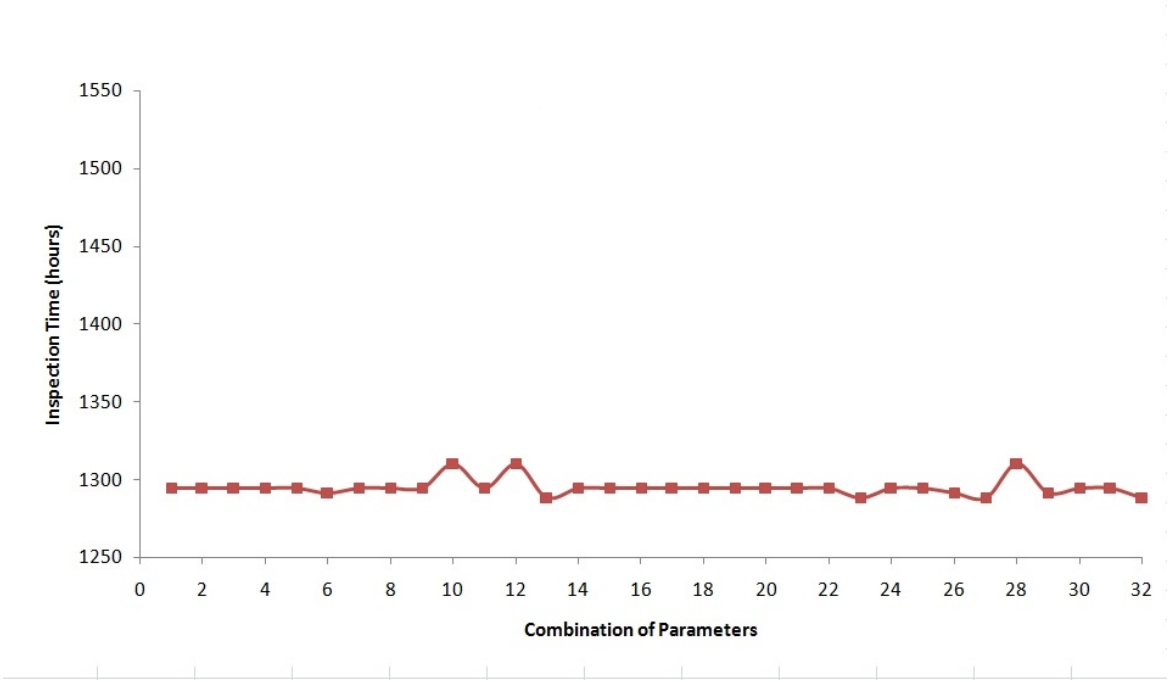


Fig. 2.8 City of Patras -Inspection time achieved per combination of the parameters

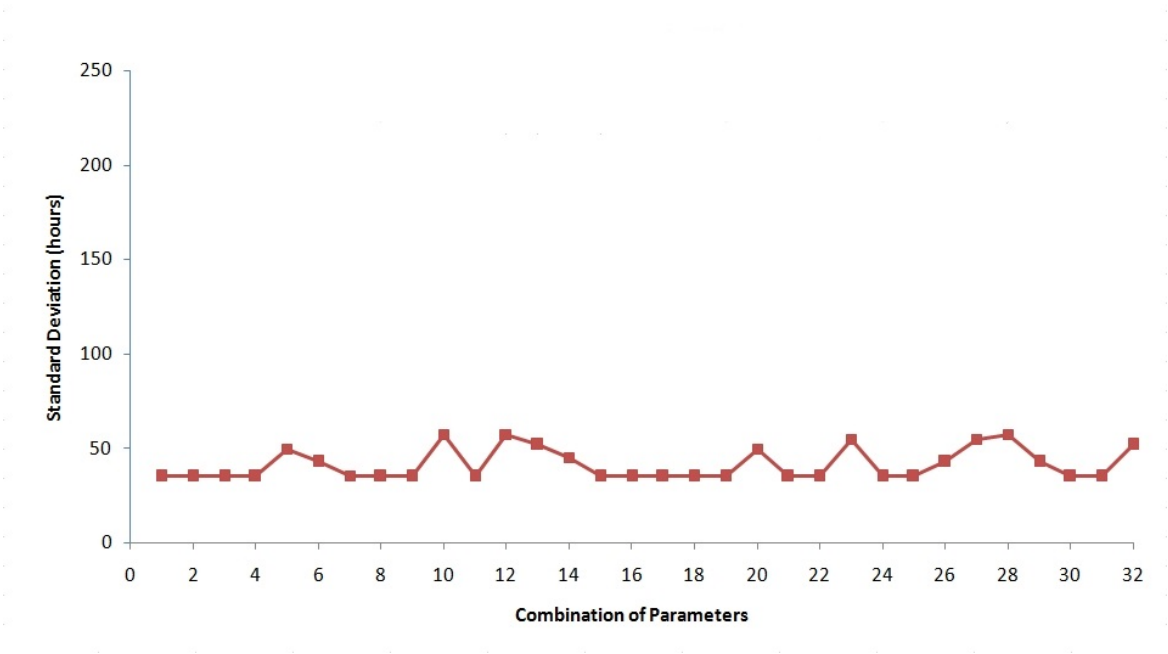


Fig. 2.9 City of Patras - Standard deviation between the inspection crews

pure random search are examined with reference to their performance when implemented to solve the districting problem of the city of Patras. In particular IHS, HS, PSO and DE algorithms are applied and compared in two problems (A and B) with 5 and 15 inspection

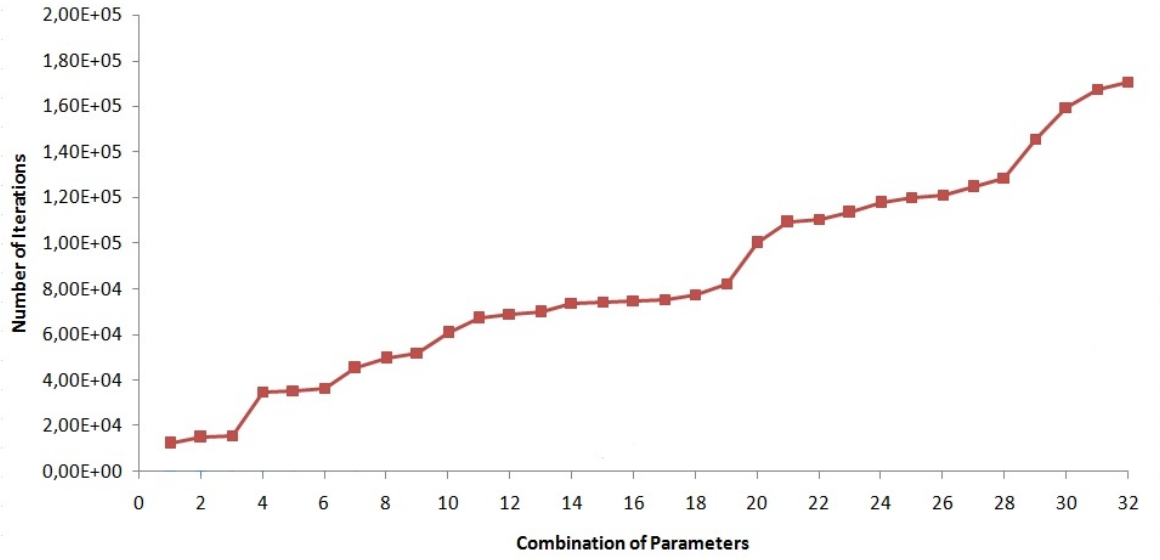


Fig. 2.10 City of Patras - Function evaluations for achieving best solution per combination of the parameters

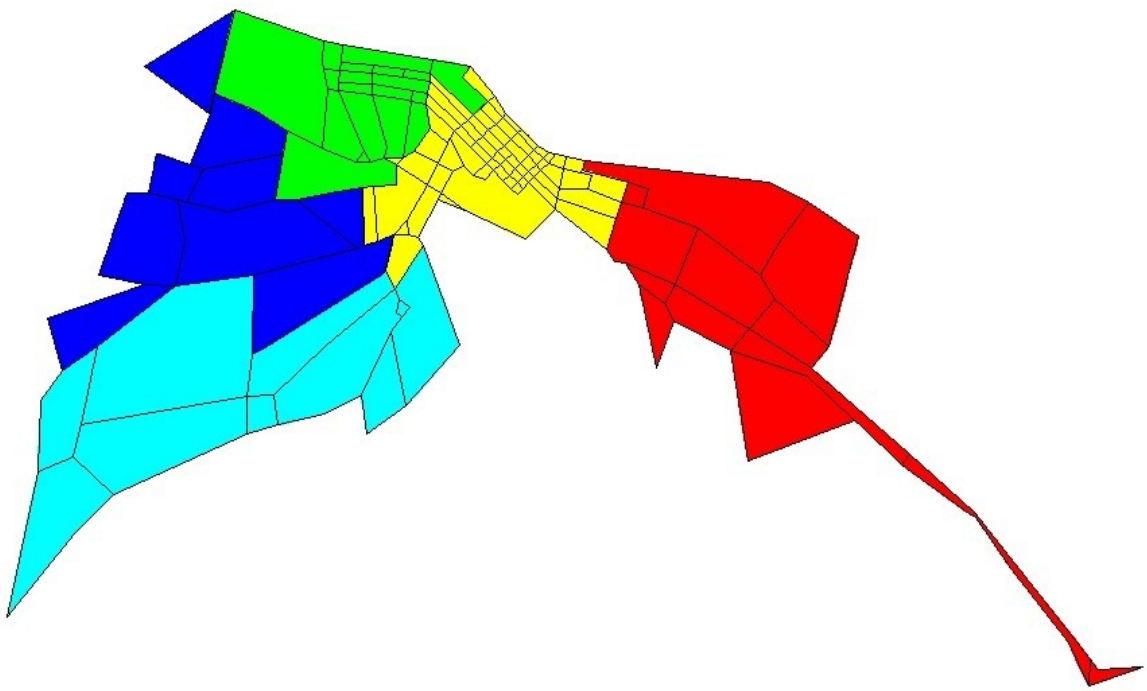


Fig. 2.11 City of Patras - Optimal districting solution proposed by IHS (Case A)

crews. A random search procedure is also implemented and compared with the above mentioned algorithms. The objective function used for all algorithms is the one used in the formulation of Eq. 2.38. The performance of the metaheuristics is influenced by the selection of their parameters. In order to study the influence of the parameters for each metaheuristic algorithm, 32 combinations of the parameters are generated by means of LHS. The resulting optimization runs for dealing with the districting problem considered for defining the parameters of the four metaheuristics and the random search procedure are equal to 5 algorithms × 32 combinations = 160 optimization runs. The termination criterion used is the maximum number of function evaluations that was set equal to 200,000 for all algorithms.

The parameters that are identified for each algorithm are: (i) For PSO, the number of particles NP defined in the range of [50, 200], the inertia weight w defined in the range of [0.01, 0.7], while the cognitive parameter c1 and social parameter c2 both defined in the range of [0, 1], the combinations considered are provided in Table 2.17. (ii) For DE, the population size NP defined in the range of [50, 200], the probability CR, the constant F and the control variable λ all defined in the range of [0, 1], the combinations considered are provided in Table 2.17. In order to ensure the properness of the parameter values used for all the algorithms mentioned before, the combinations of parameters used are generated with the use of LHS technique. The use of LHS ensures that every area of the sample space of each parameters will definitely be sampled. LHS generates 32 different sets of the parameters of each algorithm, one for every test run. Similar to the first part of the study performed for the city of Patras, the maximum time required between the inspection crews is considered as the necessary time for completing the inspection of the entire city. In the first problem examined (Problem A), 5 crews are considered for the inspection of the city of Patras. Figure 2.12 depicts the time required for the inspection of the building blocks of the city for each algorithm.

The difference between the working hours required for performing the inspection in the 32 different solutions for each algorithm is used in evaluating the quality of the results. In Table 2.18 the maximum, minimum, average, standard deviation and the coefficient of variation (COV) of the required time is depicted. Although DE is slightly better with reference to the average inspection time, as it can be seen IHS is not sensitive to the parameters of the algorithm since it always converges to almost the same optimum solution, contrary to the other algorithms where significant differences between the optimum solutions are observed. The inspection time for IHS varies from 1288.55 to 1310.44 hours with an average value of 1295.01 hours, the standard deviation is equal to 5.45 and COV is equal to 0.42%. For the case of HS, the inspection time varies from 1249.42 to 1298.97 with an average value

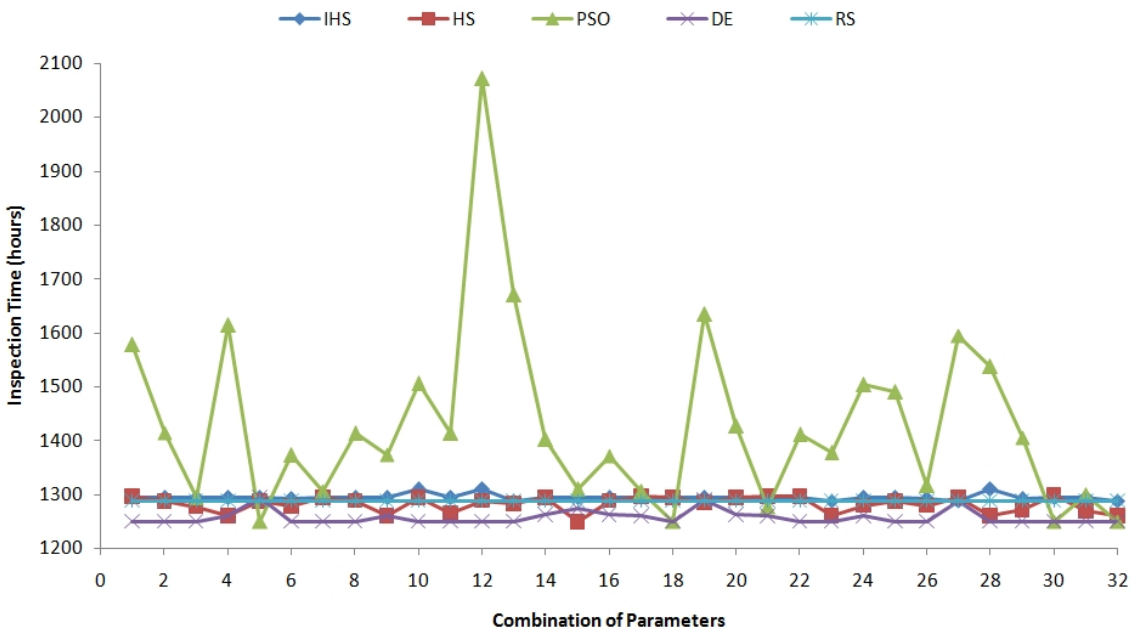


Fig. 2.12 City of Patras - Optimal districting solution proposed by IHs

of 1282.12 hours, the standard deviation is equal to 14.08 and COV is equal to 1.10%.For the case of PSO, the inspection time varies from 1250.17 to 2071.43 with an average value of 1428.25 hours, the standard deviation is equal to 168.60 and COV is equal to 11.81%. For the case of DE, the inspection time varies from 1250.17 to 1294.94 with an average of 1257.66 hours, the standard deviation is equal to 12.57 and COV is equal to 1.00%.

In the second problem examined (Problem B), 15 crews are considered for the inspection of the city of Patras. Figure 2.13 depicts the time required for the inspection of the building blocks of the city for each algorithm.

Table 2.18 Patras test case - Statistical analysis of IHS, HS, DE, PSO algorithms (Problem A – 5 Crews)

	IHS	HS	DE	PSO
Minimum Time (h)	1288.55	1249.42	1250.17	1250.17
Maximum Time (h)	1310.44	1298.97	1294.94	2071.43
Average Time (h)	1295.01	1282.12	1257.66	1428.25
Standard Deviation (h)	5.45	14.08	12.57	168.6
COV (%)	0.45	1.10	1.00	11.80

As it can be seen in Table 2.19, although DE is also slightly better compared to the other metaheuristics with reference to the average inspection time, similar to problem A, IHS is

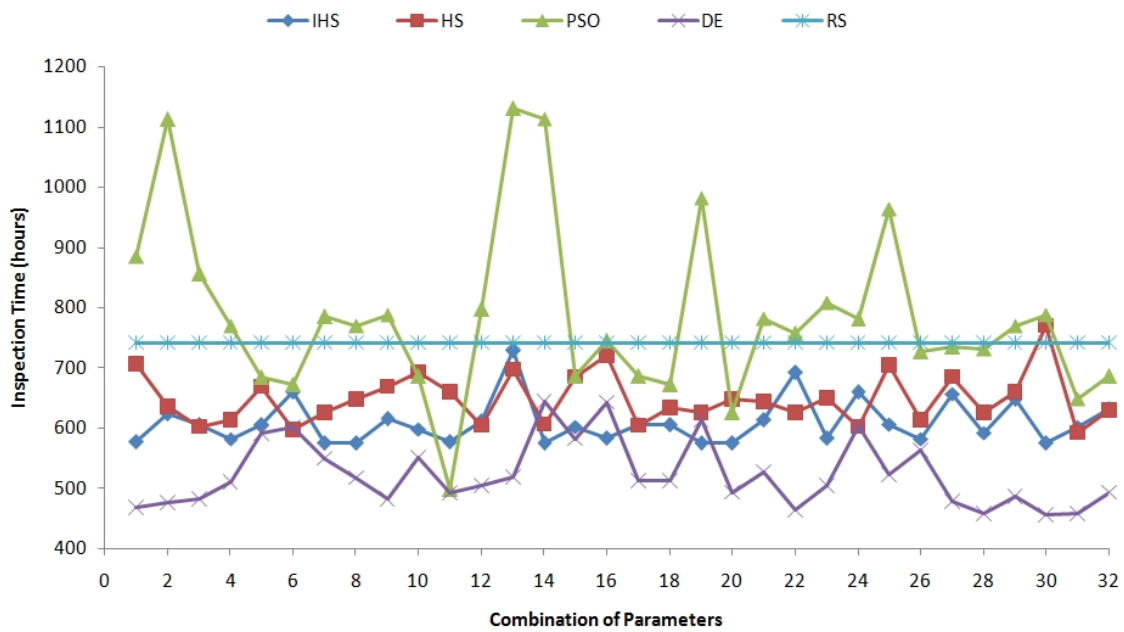


Fig. 2.13 City of Patras - Optimal districting solution proposed by IHS

not sensitive to the parameters of the algorithm as it concludes to almost the same optimum solution. The other algorithms present significant differences between the optimum solutions achieved for the 32 independent runs. The inspection time for IHS varies from 574.17 to 728.29 hours with an average value of 608.77 hours, the standard deviation is equal to 37.49 and COV is equal to 6.16%. For HS, the inspection time varies from 591.66 to 769.62 with an average value of 647.89, the standard variation is equal to 42.37 and COV is equal to 6.54%. For the case of PSO, the inspection time varies from 497.77 to 1130.73 with an average value of 784.60, the standard variation is equal to 143.13 and COV is equal to 18.24%. For the case of DE, the inspection time varies from 455.010 to 643.51 with an average value of 523.23, the standard variation is equal to 54.87 and COV is equal to 10.49%. Worth mentioning also that although the basic HS is slightly better compared to IHS with respect to the average inspection time for the case of five inspection crews, when the complexity of the problem increases (i.e. for the case of the 15 inspection crews) the superiority of the proposed IHS algorithm is obvious.

For the purposes of this tests a personal computer that consists of the Intel Core 2 Quad Q6600 2.4 GHz with 4 physical cores was used. In the last part of this numerical investigation the computational cost of the metaheuristics is examined for 200,000 function evaluations. As it can be seen in Table 2.20 PSO requires more computing time compared to all other metaheuristics for both Problems A and B while HS and IHS require almost the same time

Table 2.19 Patras test case - Statistical analysis of IHS, HS, DE, PSO algorithms (Problem B – 15 Crews)

	IHS	HS	DE	PSO
Minimum Time (h)	574.17	591.66	451.01	497.77
Maximum Time (h)	728.29	769.62	643.51	1130.73
Average Time (h)	608.78	647.89	523.23	784.60
Standard Deviation (h)	37.49	42.38	54.87	143.13
COV (%)	6.16	6.54	10.49	18.24

for both problems, while the computing time required for solving Problem B is increased by 10% compared to Problem A.

Table 2.20 Computational time for solving the two problems (in seconds)

Method	Problem A	Problem B
PSO	354.99	383.85
DE	190.32	210.36
HS	123.36	135.05
IHS	122.91	135.70

Post-Disaster management in the city of Thessaloniki

Thessaloniki is composed by $N_{SB} = 471$ grouped building blocks with various areas and built-up percentages (see Figure 2.14). For this test case two different implementations are examined. In the first one, IHS (districting problem of Eq. 2.38) is implemented while in the second one, HS is implemented on the same problem. Due to the variance between the inspection time required by the available crews, the maximum time required by the inspection crews is also considered as the necessary time for the inspection of the entire city to finish. In the test case examined, 20 crews are considered for inspecting the city of Thessaloniki. Assuming that the crews work into two 8-hour shift work-groups, a total of 10 crews are available for 16 hours inspection per day.

Figure 2.15 depicts the time required for the inspection of the building blocks of the city for each algorithm. These results are used for evaluating the performance of the algorithms. The quality of results is indicated by the difference between working hours required for completing the inspection for the 32 independent runs performed for each algorithm. As can be seen in Table 7, IHS is not sensitive to the parameters of the algorithm since it converges to almost the same optimum solution. On the other hand, HS presents significant differences

Fig. 2.14 City of Thessaloniki- Subdivision into building blocks

between the optimum solutions achieved. The inspection time for IHS varies from 5478.61 to 6034.62 hours with an average value of 5678.24 hours, the standard deviation is equal to 154.29 and COV is equal to 2.72%. For the case of HS, the inspection time varies from

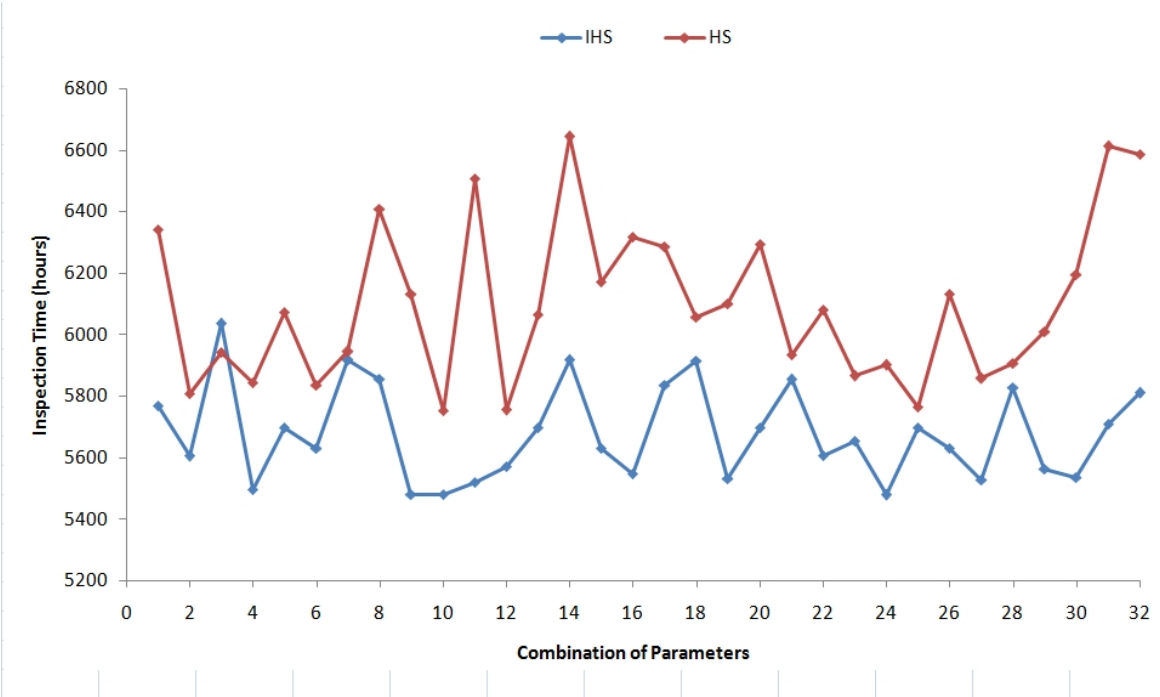


Fig. 2.15 City of Thessaloniki-Minimum inspection time achieved for each independent run

Table 2.21 Thessaloniki test case - Statistical analysis of IHS and HS algorithms (10 Crews)

	IHS	HS
Minimum Time (h)	5478.61	5751.64
Maximum Time (h)	6034.62	6643.47
Average Time (h)	5678.24	6097.35
Standard Deviation (h)	154.29	260.34
COV (%)	2.72	4.27

5751.64 to 6643.47 with an average value of 6097.35, the standard variation is equal to 260.34 and COV is equal to 4.27%. This larger-scale test case and the results presented in Figure 2.16 show that IHS can achieve significantly better results compare to HS when dealing with the districting problem. It can also be seen that IHS produces results of lesser variation than HS as it appears to be less sensitive to its parameters. This confirms the finding of the Patras test example that when the complexity of the problem increases the superiority of the proposed IHS algorithm becomes obvious.

The above mentioned IHS, PBA and their tests have been published in journals [94, 93] after being reviewed and part of the document is according to the published one.

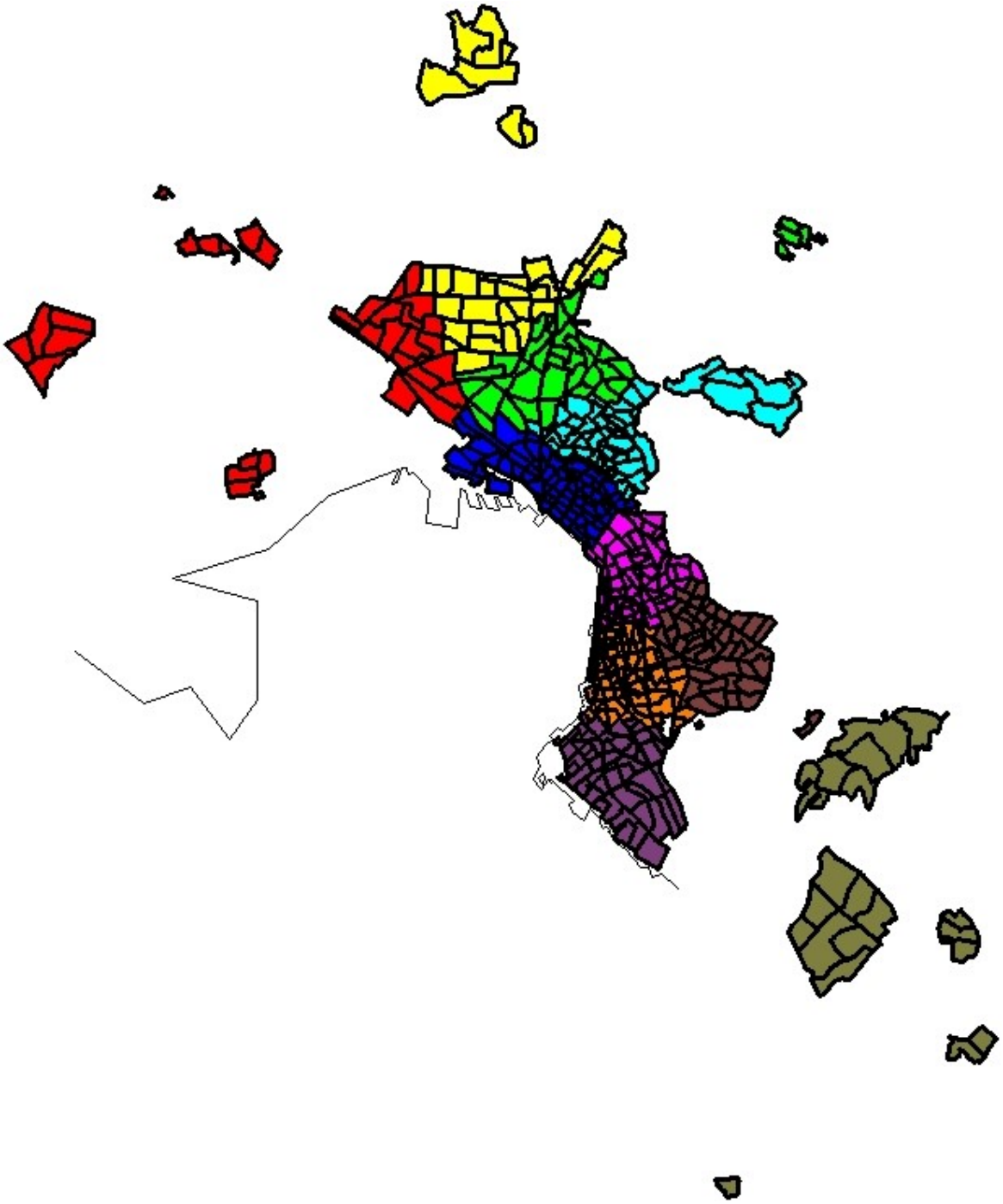


Fig. 2.16 City of Thessaloniki - Subdivision into inspection areas

Chapter 3

Deep Learning methodologies

Machine learning (ML) techniques can be described as procedures that allow computers to make decision and act according to rules defined by "correct" decisions or actions while adapting to successful or unsuccessful results of previously made decisions in an iterative manner [137]. The basic principles of ML can be traced in combined disciplines of biology, mathematics and physics. The main usage of ML techniques was to exploit data-sets and produce results according to data included in that set. Nowadays as production, usage and storage of data grows continuously, interest on such methods also increases accordingly and it expands on several new sectors. Some of the sectors that ML is used are health care, data mining, computer vision, autonomous driving, risk assessment, natural language processing, fraud detection, investment strategies, engineering, etc. Examples of such applications are medical diagnosis [110], big data exploitation [133], visual recognition [41], visual perception in driving [27], landslide susceptibility [166], sentiment analysis [135], network intrusion detection [197], FOREX market prediction [149], structural damage detection [23] and failure diagnosis [207]. It is also worth mentioning that ML techniques are already used in many everyday applications used by most people without even knowing. Email spam filtering, auto-reply and digital personal assistants are ML-based applications while chat-bots for on-line customer support and route selection along with estimated time of arrival in GPS systems are also ML-driven. Search engine results and personalized adds in web-pages are also performed with ML procedures. From an engineer's point of view, AI-assisted drone-based 3D modeling and site planning is already at use, energy consumption predictions with the help of ML are performed by energy companies while also, structural health monitoring with ML usage are already a reality as well.

3.1 Machine learning: Types, Methods & Problems

Machine learning can be categorized in many ways according to several categorization criteria. If the criterion is the learning procedure used, all ML algorithms fall into one of the categories listed bellow [137]:

- Supervised learning,
- Unsupervised learning,
- Semi-supervised learning,
- Reinforcement learning and
- Evolutionary learning

In supervised learning, algorithms are calibrated (trained) on a dataset with an already known correct input-output mapping. Based on the achieved "experience", the algorithm can asses the output of a not-before-seen input. The training dataset can be described as follows:

$$Training\ data = \left[\begin{array}{cccc|ccc} x^I_{1,1} & x^I_{1,2} & \dots & x^I_{1,n} & x^T_{1,1} & \dots & x^T_{1,m} \\ x^I_{2,1} & x^I_{2,2} & \dots & x^I_{2,n} & x^T_{2,1} & \dots & x^T_{2,m} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x^I_{d,1} & x^I_{d,2} & \dots & x^I_{d,n} & x^T_{d,1} & \dots & x^T_{d,m} \end{array}\right] \tag{3.1}$$

where d is the population of the n-dimensional input vectors I and the m-dimensional output vectors T. Some of the most common problems where supervised learning is applied are regression and classification problems.
Unsupervised learning is used for discovering patterns in input data. There exists no output data and this type of algorithms are used to match input vectors of similar characteristics. An expression of unsupervised learning can be formulated as follows:

$$\forall \{X^I\} \in [S] = \begin{bmatrix} x^I_{1,1} & x^I_{1,2} & \dots & x^I_{1,n} \\ x^I_{2,1} & x^I_{2,2} & \dots & x^I_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x^I_{d,1} & x^I_{d,2} & \dots & x^I_{d,n} \end{bmatrix} \quad \exists\ C_j\ \ where\ \ \{X^I\} \in C_j \tag{3.2}$$

where $[S]$ is the input sample containing d input vectors ($\{X^I\}$), C_k are the k clusters of similar characteristics and $j \in [1,k]$. A typical case for unsupervised learning use is data clustering and also data dimensionality reduction.

Semi-supervised learning, as implied by the title, refers to the combination of supervised and unsupervised learning. Usually, a percentage of input vectors have a predefined output while the rest do not. In can be said in general that it is used in order to increase the learning grade in comparison to unsupervised learning.
Reinforcement learning can be described as almost-supervised learning. The reason behind this is that the correct target per training input vector is known but there is no exploitation of this information regarding correction of wrong estimations. The only information taken under consideration during the training phase is whether the proposed result is correct or not.
Evolutionary learning, as mentioned earlier in Chapter 2, is a procedure mimicking the evolution of species in nature. Adaptability and survival of the fittest are the main characteristics of such methods. The quality of solution represented by the particle-solution vector defines the survival or reproduction probability of the current solution.

Several machine learning methods have been proposed in the past years. Some of the most well known are [146]:

- Decision trees
- Artificial neural networks (Shallow neural networks, deep neural networks, etc.)
- Bayesian learning (Naive Bayes, bayesian belief networks, etc.)
- Instance-based learning (k-Nearest neighbour, radial basis functions, etc.)
- Kernel methods (Support vector machines (SVM), etc.)
- Metaheuristics (Genetic algorithm, particle swarm optimization, etc.)

Decision tree classifiers [177] are commonly used as decision making applications. The basic idea behind this method is to divide the decision into a graph of smaller and less complicated decisions and finding the correct by following a series of answers of these smaller decisions. A decision tree is designed once the structure is defined, a sub-problem and a decision strategy is assigned to each node. A training procedure is needed before applying a decision tree methodology on a specific problem.
Artificial neural networks (ANN), firstly introduced in 1943 [139] are computational models, originally inspired by the functionality procedure of neurons in the brain. This first ANN model was reading a set of input x, calculated their weighted sum and returned as output,

$v(x)$, a value equal to 0 or 1 according to:

$$v(x_t) = \begin{cases} 1 & if\ x_t \geq 0 \\ 0 & otherwise \end{cases}$$
$$where : x_t = \sum_j w_{i,j} * x_{j(t-1)} - \mu_i \tag{3.3}$$

where i, j are neurons, $w_{i,j}$ is the connection weight of the two neurons and μ_i is the threshold value for neuron i. From simple binary threshold unit to up-to-date complex networks inspired by visual cortex [119], many different implementations of ANN have been proposed in various fields. Simple perceptrons, multi-layer networks, recurrent networks are some of the various ANN proposed [66]. ANN are implemented through training on data according to a case-study.

Bayesian learning uses preexisting "experience" along with data observation in the learning procedure of defining probabilities of hypothetical output signals [146]. According to Bayes theorem, the posterior probability of a hypothetical output u based on training data TD is calculated as follows:

$$P(u|TD) = \frac{P(TD|u) * P(u)}{P(TD)} \tag{3.4}$$

Bayesian learning methods are usually used as classifiers in ML due to their probabilistic nature.

Instance-based learning takes advantage of memory-stored training instances to handle learning procedures [2]. This methodology is applied on supervised learning tasks. Each training instance can be defined by a set of m parameters. By examining differences in these parameters between a training data input I and a testing data input T, similarity between I and T can be calculated and T can be properly classified. Evaluation on performed classifications can be used for upgrading algorithmic performance in future classifications. Ever since their first appearance [144], they have been applied to several classification problems [32] while plenty variations have also been proposed [18, 218]. Kernel methods and their main representative were firstly introduced in 1995 [33]. They are mainly used in supervising learning methodology for creating relationships between input and output data by taking advantage of inner products of vectors in higher dimensional spaces in comparison to the dimensions of the input space [188]. Metaheuristic approaches on machine learning were presented in detail in Chapter 2 of this thesis.

3.2 Deep Learning

Deep learning (DL) methods, also known as hierarchical learning methods can be considered as an evolution of neural networks and their architectures [39]. Shallow neural networks, ever since their first appearance [139], have been used in several applications for non-linear transformation of data and discovering relationship between reason (input) and result (output). Inevitably, as models became larger and more complex, the use of shallow architectures was inefficient while also, training of multi-layer structures was not possible [185] at least up to 2006 [75, 76, 164] where new architectures in combination with new training techniques surfaced the research scheme. While several definitions of DL methods are proposed in literature, according to LeCun et al. [117], deep learning *is the group of applications that allow computational models consisting of several processing layers to learn representations of data with multiple abstraction levels*. Goodfellow et al. [63] suggest that DL is defined according to the actual learning procedure. AI attempts to "teach" computers from gained experience without the need of human interaction and give them the ability to suggest solutions based on the hierarchy of concepts. By "understanding" simpler concepts, knowledge can be transferred to more complex ones. By designing the multiple layers that formulate the above procedures it can be seen that it is a deep graph. Hence, this approach in named deep learning.

The generality and architectural characteristics of DL methods allow them to be applicable to all machine learning problem types as almost all learning type categories (as described earlier) are covered by them. Actually, in many modern problems, DL methods are considered and proven as the most successful methodologies. It is worth mentioning though that different type of problems may require handling by a separate DL approach as not all methods present identical advantages and disadvantages. This necessity is also amplified due to different architecture of DL methods.

The main categories of deep learning methods, based on network architecture differentiation are the following [156]:

- Unsupervised pre-trained networks
- Convolutional neural networks
- Recurrent neural networks
- Recursive neural networks

As mentioned previously, deep architectures could not be trained prior to 2006. According to the work presented then by Bengio et al. [14], Hinton et al. [75], Poultney et al. [164],

training of deep architectures could be achieved by a separating the training phase into two sub-phases. Initially, a greedy layer-wise unsupervised pre-training is performed followed by the final phase of supervised training. With the use of this technique, all layers of the network are separated in pairs as it can be seen in Fig. 3.1 and trained in sequential order. In Fig. 3.1, a deep neural network of $m+1$ layers is divided in m pairs with the first and final layer having s and q nodes respectively.

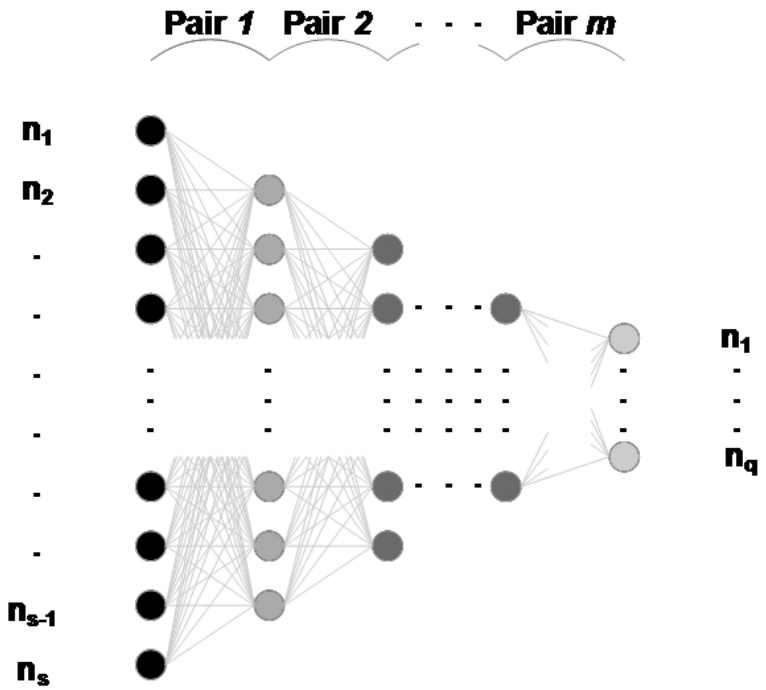


Fig. 3.1 Pair separation of a deep network for unsupervised pre-training.

Once this step is completed, the whole deep structure is trained or actually fine-tuned in a supervised manner with the use of available database as it can be seen in Fig. 3.2 where the non-linear correlation between s vectors named d of size n (input) and s vectors named t of size z (output) are defined by the deep network consisting of $m+1$ layers L. In general, it can be said that the pre-training phase assists greatly in the optimization of the performance of the deep network by offering a better starting point for the optimization procedure of the supervised training set phase [46]. All unsupervised pre-training networks are formulated under this architecture. Some of the most widely known methodologies of this type are deep belief networks and autoencoders. Deep belief networks are thoroughly presented in the following section of this chapter.

Convolutional neural networks (CNNs), where firstly introduced by LeCun et al. [119] in 1989, inspired by the structure and functionality of the visual cortex of animals and are mainly used for image data studies. A detailed presentation of CNNs can be viewed in the following sections.

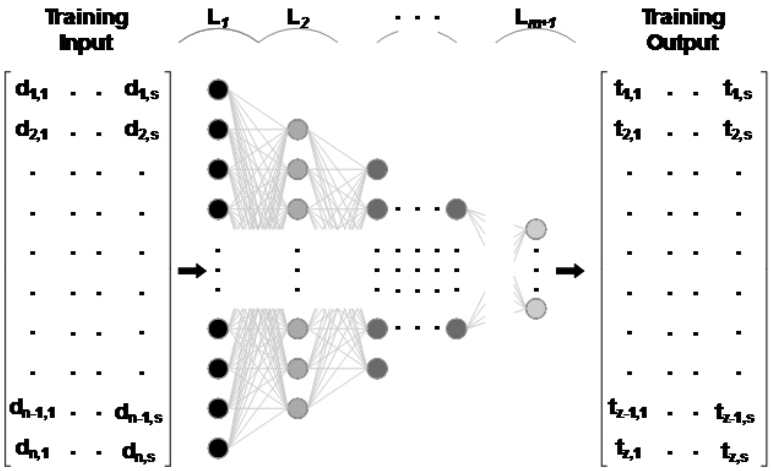


Fig. 3.2 Supervised training of the deep network.

Recurrent neural networks (RNNs) were firstly introduced around 1990 [175, 184] as a modified version of feedforward networks. The difference lies in the fact that RNNs introduced the application of time-steps in the network in order to be able to exploit sequential information. It can be said that RNNs added an extra dimension to feedforward networks as a node in an RNN is a "folded" value gained after training on several time-steps. This structure can be seen in Fig. 3.3 where apart from an RNN, an unfolded node in the z direction is presented, d_t is the input value of a timeseries with 3 steps and q_t is the output for each time-step. It can be seen that the finally proposed output takes under consideration "experience" from neighbouring values of time-steps. RNNs, from an architectural point of view, are ideal for handling problems with time-depending data as they are trained on sequential input data and propose an output based on the time-sequence of the input and not just the current value, by utilizing memory gained from the previous calculations.

Recursive neural networks (RcNNs) were firstly introduced in 1990 [162]. RcNNs are similar to recurrent neural networks as both have the ability to handle inputs of different sizes. Contrary to RNNs, RcNNs present a tree-shaped structure where the base represents inputs and the top represent outputs [156]. The architecture of the networks tree is the basic problem when applying RcNNs on a problem and several approaches have been proposed regarding this subject. Usually, the formation of the structure of the RcNN depends of the

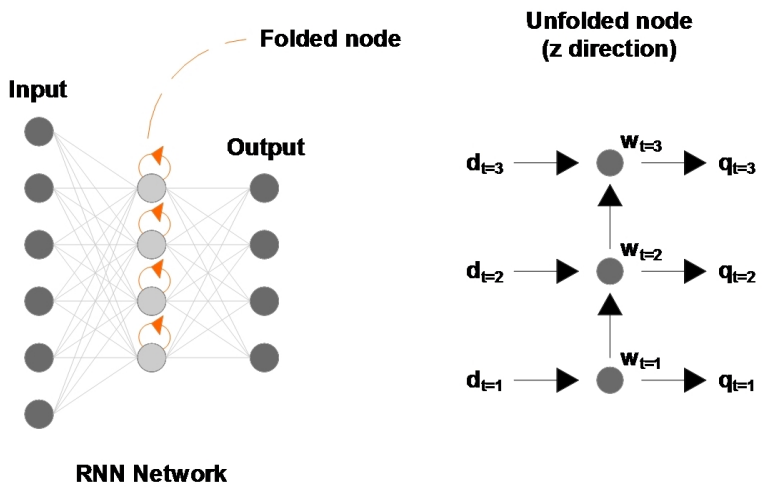


Fig. 3.3 Recurrent neural network and unfolded node.

nature of the problem and the data structure available. RcNNs are mainly used in the fields of natural language processing and computer vision.

3.2.1 Restricted Boltzmann Machines

In this part of the thesis, a detailed presentation of Restricted Boltzmann Machines (RBMs) can be found.

RBM architecture

RBMs where initially developed in 1986 by Smolensky [196] as "harmonium". Originating from Boltzmann Machines (BM), they can be described as energy-based probabilistic graphical models while they can also be considered as stochastic neural networks [49, 71]. Boltzmann machines (BMs) were initially introduced around 1985 [1, 48, 77]. Similar to Hopfield networks[79], BMs are networks consisting of fully and symmetrically connected nodes containing binary stochastic units [181] which behave as neural nets activators with on/off signals. Boltzmann machines also belong to the class of energy-based models. Energy-based models are used for defining correlations between sets of data-variables. This is achieved by using an energy function for evaluating the quality of correlations with respect to correct or not output. Through training, the energy function is constantly being reevaluated in order to ensure that correct outputs will have lower energy while wrong ones will have higher one [120]. A BM network consists of two layers where the first one, named visible v consists of m nodes and the second one called hidden h consisting of k nodes. As stated earlier, all nodes of both v and h layers are connected with each other as seen in Fig. 3.4

where P_1, P_2 and P_3 are the characteristics (weights and biases) of connections between nodes only in v, only in h and connections between nodes belonging in different layers respectively. For a given state of the above network can be described as follows:

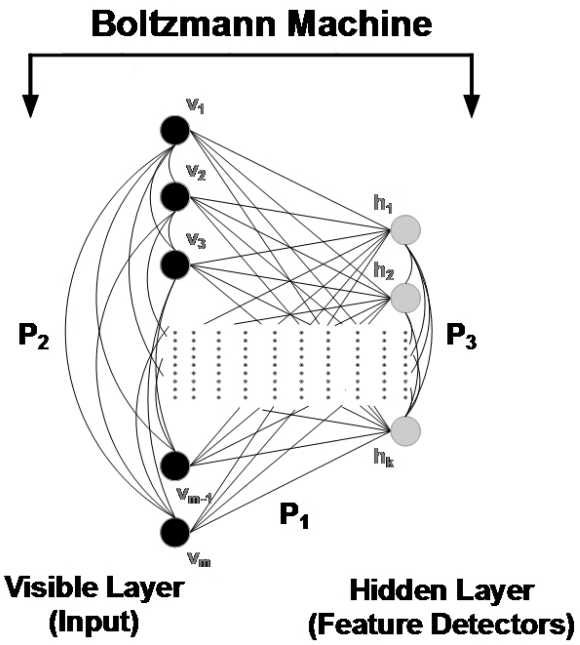


Fig. 3.4 Boltzmann Machine representation.

$$E(v,h;\theta) = -v^T P_1 h - \frac{1}{2} v^T P_2 - \frac{1}{2} h^T P_3 \tag{3.5}$$

where θ represents the defined values of P_1, P_2 and P_3, v is the input vector and h is the output vector. As BMs are probabilistic models, they focus on defining a distribution over v and model that distribution in another (higher or lower) order with the use of feature detectors h. The probability distribution [131] of the above model is defined as:

$$P(v,h;\theta) = \frac{e^{-E(v,h;\theta)}}{Z} \tag{3.6}$$

where Z is a normalization factor, also known as partition factor and is defined as follows:

$$Z = \sum_v \sum_h e^{-E(v,h;\theta)} \tag{3.7}$$

The BM can be trained with the use of stochastic gradient descent on the negative log-likelihood of data but due to its architecture and populations of connections, deep networks present very low training speeds and in some cases very low success rates [72].

These difficulties inspired the creation of RBMs which are almost identical in terms of architecture and functionality to BMs. The "restricted" part describes the fact that contrary to BMs, connections exist only between nodes that belong to a different layer.As RBMs can be considered as a subcategory of Markov Random Fields, they consist of two layers where the first one, named visible v consists of m nodes and the second one called hidden h consisting of k nodes and each node belonging to one of the two, v and h, layers is connected to all nodes belonging to the other layer as seen in Fig. 3.5 and in comparison to Fig. 3.4. P_1 are the characteristics (weights and biases) of existing connections between nodes. The parameters

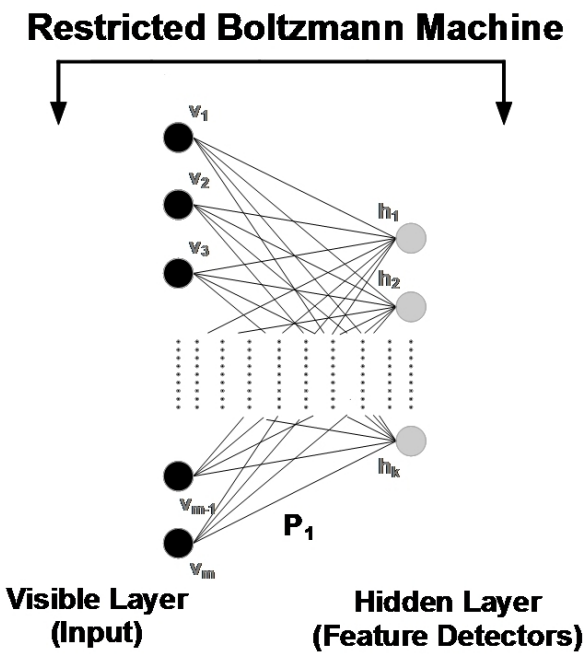


Fig. 3.5 Restricted Boltzmann Machine representation.

P of the network include the connection weights $w_{i,j}$ where $i \in [1,m]$ and $j \in [1,k]$, the biases b_i of the nodes of the visual layer and the biases c_j of the hidden layer's nodes. As BMs, RBMs are also probabilistic graphical models and an energy function expresses the quality of a state of the network. As input data can be either binary or real valued, this function changes as the distribution used changes from Bernoulli to Gaussian respectively [76]. The most commonly used types of RBMs are Bernoulli-Bernoulli RBM (BBRBM) and Gaussian-Bernoulli RBM [222]. Due to non existing connections between nodes of the same layer in RBMs, the energy function described in Eq. 3.5 in transformed to:

$$E(v,h;\theta) = -b^T v - c^T h - v^T W h \Rightarrow \tag{3.8}$$

$$E(v,h;\theta) = -\sum_{i=1}^{m} b_i v_i - \sum_{j=1}^{k} c_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{k} w_{ij} v_i h_j \tag{3.9}$$

in the case that a BBRBM. In the case of GBRBM, the energy function is described as follows [39]:

$$E(v,h;\theta) = -\frac{1}{2}\sum_{i=1}^{m}(b_i - v_i)^2 - \sum_{j=1}^{k} c_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{k} w_{ij} v_i h_j \tag{3.10}$$

The Gibbs distribution expresses the joint probability distribution for the current model with the use of Eq. 3.6. As no connections exist between nodes of the same layer:

$$p(h|v) = \prod_{j=1}^{k} p(h_j|v) \tag{3.11}$$

and

$$p(v|h) = \prod_{i=1}^{m} p(v_i|h) \tag{3.12}$$

As a result, the conditional probabilities for the case of BBRBM (3.13,3.14) and GBRBM (3.15,3.16) can be expressed as follows:

$$p(1|v;\theta) = \sigma(\sum_{i=1}^{m} w_{ij} v_i + c_j) \tag{3.13}$$

$$p(1|h;\theta) = \sigma(\sum_{j=1}^{k} w_{ij} h_j + b_i) \tag{3.14}$$

$$p(1|v;\theta) = \sigma(\sum_{i=1}^{m} w_{ij} v_i + c_j) \tag{3.15}$$

$$p(1|h;\theta) = N(\sum_{j=1}^{k} w_{ij} h_j + b_i, 1) \tag{3.16}$$

where $\sigma(x) = \frac{1}{1+exp(-x)}$ is the logistic function and N is the Gaussian $N(m,v)$ with m being the mean and v the variance.

Training RBM with contrastive divergence

The most common method for training RBMs is based on applying gradient ascent on the log-likelihood. Since RBMs can be considered as Markov Random Fields, the log-likelihood

of an RBM similar to the one presented in Fig. 3.5 can be expressed as follows [50]:

$$ln[\mathcal{L}(\theta|v)] = ln[p(v|\theta)] \tag{3.17}$$

where the marginal distribution of v is calculated as follows:

$$p(v|\theta) = \sum_{h=1}^{k} p(v,h) = \frac{1}{Z}\sum_{h=1}^{k} e^{-E(v,h)} \tag{3.18}$$

where $Z = \sum_{v,h} e^{-E(v,h)}$. Accordingly, Eq. 3.17 is transformed:

$$ln[p(v|\theta)] = ln(\frac{1}{Z}\sum_{h=1}^{k} e^{-E(v,h)}) = ln(\sum_{h=1}^{k} e^{-E(v,h)}) - ln(\sum_{v,h} e^{-E(v,h)}) \tag{3.19}$$

The gradient of the log-likelihood can be calculated as follows:

$$\begin{aligned} \frac{\partial ln\mathcal{L}(\theta|v)}{\partial\theta} &= \frac{\partial(ln\sum_{h=1}^{k} e^{-E(v,h)})}{\partial\theta} - \frac{\partial(ln\sum_{v,h} e^{-E(v,h)})}{\partial\theta} \Rightarrow \\ \frac{\partial ln\mathcal{L}(\theta|v)}{\partial\theta} &= -\sum_{h=1}^{k} p(h|v)\frac{\partial E(v,h)}{\partial\theta} + \sum_{v,h} p(v,h)\frac{\partial E(v,h)}{\partial\theta} \end{aligned} \tag{3.20}$$

It can be witnessed that the quantities on the right part of the equation are in accordance to the theorem [21] that the derivative of the log-likelihood of RBM input with respect to parameters is expressed as:

$$\frac{\partial p(v,h)}{\partial\theta} = -\left\langle \frac{\partial logE(v,h)}{\partial\theta} \right\rangle_0 + \left\langle \frac{\partial logE(v,h)}{\partial\theta} \right\rangle_\infty \tag{3.21}$$

where the first part is an average, over v and h, with respect to RBM input distribution multiplied with $p(h|v)$ and the second part is an average with respect to the distribution of RBM output [116]. The calculation of the second part is extremely computationally heavy. It can be proven that the partial derivatives of Eq. 3.20 are defined as follows [50]:

$$\frac{\partial ln\mathcal{L}(\theta|v)}{\partial w_{ij}} = p(1|v)v_i - \sum_{i=1}^{m} p(v)p(1|v)v_i \tag{3.22}$$

$$\frac{\partial ln\mathcal{L}(\theta|v)}{\partial b_i} = v_i - \sum_{i=1}^{m} p(v)v_i \tag{3.23}$$

$$\frac{\partial ln\mathcal{L}(\theta|v)}{\partial c_j} = p(1|v) - \sum_{i=1}^{m} p(v)p(1|v) \tag{3.24}$$

As calculations of the second parts of Eq. 3.22-3.24 are significantly computationally heavy, Gibbs sampling is used for generating samples of the expectations used for approximation [75]. In this methodology, known as contrastive divergence (CD) [70, 71], instead of producing a large number of samples, k samples are generated. According to CD, the gradient of the log-likelihood is approximated as follows [50]:

$$CD_k(\theta, v_0) = -\sum_{h=1}^{k} p(h|v_0)\frac{\partial E(v_0,h)}{\partial \theta} + \sum_{h=1}^{k} p(h|v_k)\frac{\partial E(v_k,h)}{\partial \theta} \tag{3.25}$$

The above equations concern the binary RBM. The same procedure can also be implemented for real-valued RBMs [217]. It has been proven in theory and actual applications that even for $k = 1$, the results are more than satisfying [14, 76, 179] which accelerates the gradient estimation even more.

3.2.2 Deep Belief Networks

Deep belief networks (DBNs) are actually considered as the pioneer model of deep learning practices as, when firstly introduced [76, 73], they were the first deep model that after being successfully trained, managed to achieve better results than SVMs in a classification problem [75]. DBNs are probabilistic generative models which contain a population of stochastic, latent variables. These variables are used as feature detectors of higher order correlations in training datasets.

A DBN is created when several RBMs are stacked in a sequential manner [72]. In this architecture is formed by the principle that the hidden layer of RBM_{i-1} is the visible layer of RBM_i. It is also worth pointing out that in DBNs, all layers have directed connections except for the last two which have undirected ones [63]. A representation of such a network can be seen in Fig. 3.6 where a DBN consisting of 4 RBMs is displayed. RBM1 consists of layers L1 and L2 and the rest RBMs are defined accordingly. Based on the previous description, L2 is the hidden layer of RBM1 while it is also the visual layer of RBM2.

As mentioned before, training of deep networks of such architecture was not possible until a new training methodology was proposed [75]. This methodology consists of a two-step training procedure. In the first step, called pre-training, each RBM of the DBN is trained in an unsupervised manner while in the second step, called fine-tuning, the whole DBN undergoes supervised training [74, 75]. The pre-training is performed by applying the contrastive divergence method in each RBM sequentially. For the first one, the visual layer

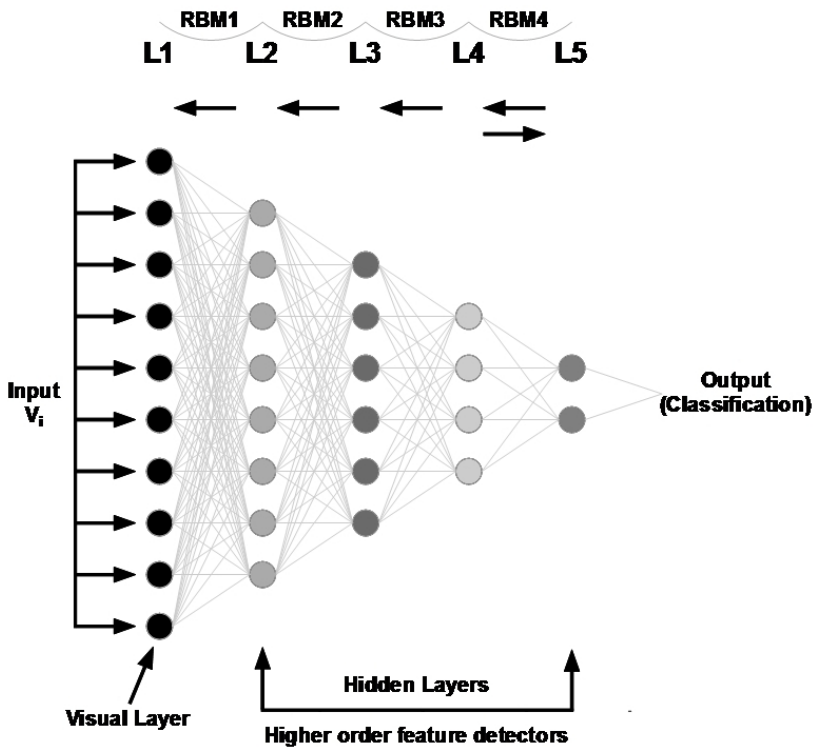


Fig. 3.6 Deep Belief Network example.

refers to the actual input while the hidden layer, once trained, will act as the visual layer in the training of RBM2 as described previously. The weights of the connections of each RBM are updated as follows:

$$\begin{aligned} \frac{\partial \log p(v;\theta)}{\partial w_{ij}} &= \langle v_i h_j \rangle_{\text{input}} - \langle v_i h_j \rangle_{\text{model}} \\ w_{ij}^{new} &= w_{ij} + e\Delta w_{ij} \end{aligned} \tag{3.26}$$

where e is a parameter defining the desired range of weight change, known as weight learning rate.

The fine-tuning is performed once pre-training is finished with the use of back propagation [176] and conjugate gradient algorithm [68]. Through this part, the weights that are proposed from the unsupervised pre-training are tuned working with the deep neural network as a whole. Back-propagation is an iterative procedure, which updates weight values according to the difference between a target output and the networks output for specific weights. CG method is usually used for adjusting the weights while steepest descent or others can also be used. For example, in the case of an RBM trained over m samples and with outputs of size n,

the weight update for all weights in the network is:

$$W_{all}^{new} = W_{all} - c\frac{\partial E}{\partial W} \tag{3.27}$$

where

$$E = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}(t_j^i - P_J^I)^2, \tag{3.28}$$

m is the number of training samples, n is the size of outputs, t is the target output, p is the generated output and c is a weight learning rate parameter.

A similar architecture of DBNs are the Deep Boltzmann Machines (DBMs), introduced in 2009 [180], which also are probabilistic graphical models with multiple layers of stochastic nodes. DBNs and DBMs share many common features in terms of architecture and functionality. where the fact that no connections exist between nodes of the same layer and all nodes

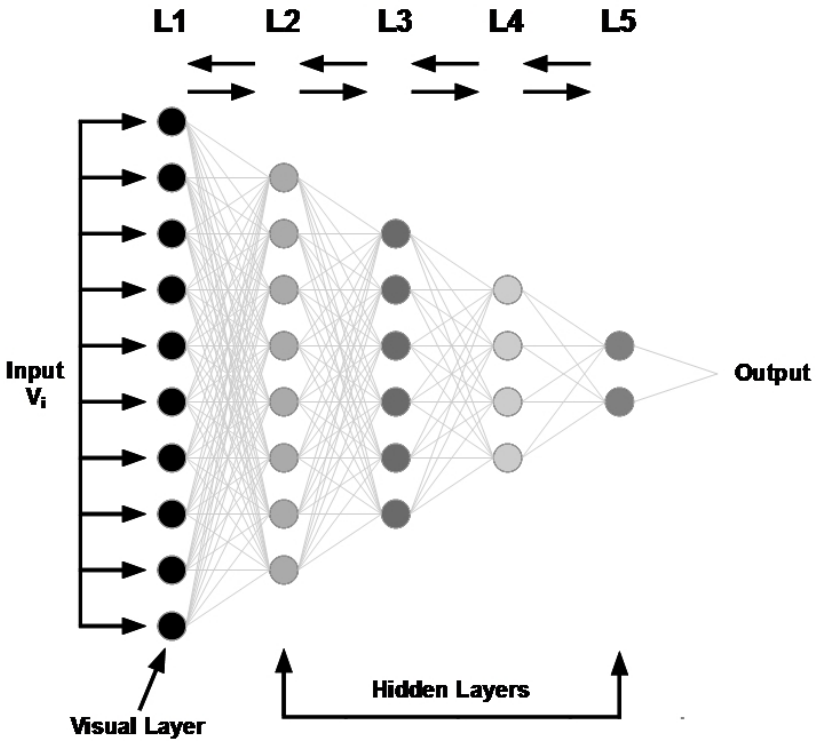


Fig. 3.7 Deep Boltzmann Machine example.

of one layer are connected to all nodes of the previous and the following layer.

The main difference is the fact that all layers of a DBMs are undirected. Apart from the first layer, also known as visible, all other layers of the network are hidden. A simple DBM with five layers can be seen in Fig. 3.7 As energy-based models, an energy function is used for defining the joint probability distribution of the variables, similarly to DBNs [63]. The

energy of a DBM like the one presented in Fig 3.7 can be calculated as follows:

$$E(v,h^1,h^2,h^3,h^4,h^5;\theta) = -v^T W^1 h^1 - h^1 W^2 h^2 - h^2 W^3 h^3 - h^3 W^4 h^4 - h^4 W^5 h^5 \quad (3.29)$$

while the probability assigned to input v is expressed as:

$$p(v;\theta) = \frac{1}{Z} \sum_{h^1:h^5} e^{-} E(v,h^1,h^2,h^3,h^4,h^5;\theta) \quad (3.30)$$

where θ represents the weights and biases of connections between layers of the network. The training of a DBM is accomplished with the use of a greedy layer-wise pre-training as in the case of DBNs by decomposing the DBM into RBMs and with using back propagation.

3.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), are probably the most well known deep learning methodology nowadays, especially in the field of computer vision. The excellent performance of CNNs is partially explained by the fact that their architecture was inspired by the functionality of the visual cortex of animals [82, 83]. Receptive fields are used for detecting features in the visual input. The first step towards CNNs was the algorithmic translation of the functionality of receptive fields as an hierarchical neural network for pattern recognition [56] in 1988. In the following years, significant work on CNNs was presented [118, 119] but it was not until many years later that CNNs attracted major attention due to their performance on image recognition [111]. Since then, several deep and complicated CNNs have been proposed with remarkable results in object recognition, computer vision, etc.. Some of these networks are LeNet 5 [121], ZFNet [226] and GoogLeNet [203].

CNN architecture and training

CNNs present a general architecture consisting of some basic layers while differentiations usually refer to the number of layers or their sequence and several in-layer functions that will be described in the following parts. A typical CNN architecture, as proposed by LeCun for LeNet, can be seen in Fig. 3.8. The basic layers in a CNN are:

- Convolutional layers
- Pooling layers
- Fully connected layers

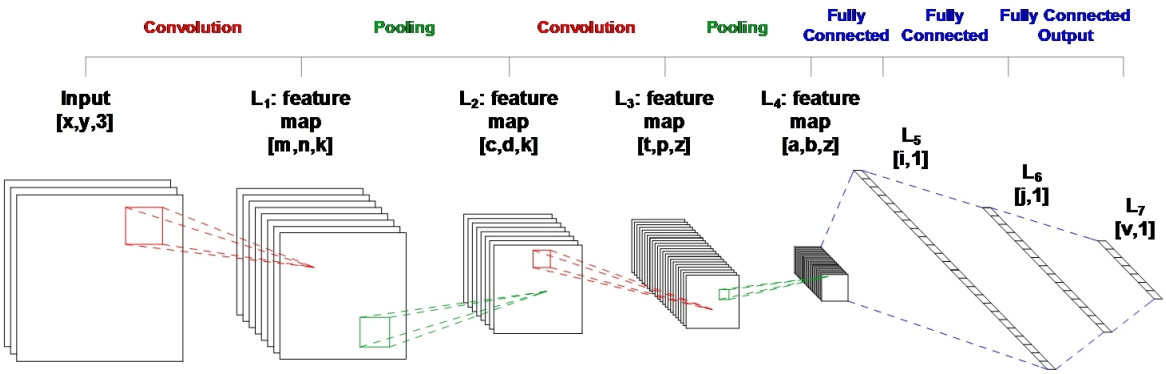


Fig. 3.8 Convolutional Neural Network architecture.

and some more rarely used like regularization layers [63].

Inputs of CNN are matrices D with $[x,y,l]$ dimensions. As mainly used for image processing, x and y represent the horizontal and vertical number of pixels respectively while l is equal to three as it represents the three colour channels R,G,B. It is worth pointing out that it is not a strict rule as CNNs can handle 2D and 1D matrices as well. In the following part of the chapter, a 2D matrix with dimensions equal to $[8,8,1]$ as seen in Eq. 3.31 will be used for explaining the various functions of CNN layers.

$$D = \begin{bmatrix} 8 & 2 & 1 & 4 & 5 & 7 & 4 & 2 \\ 3 & 4 & 5 & 1 & 7 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 & 6 & 1 & 9 & 5 \\ 7 & 9 & 9 & 7 & 5 & 3 & 3 & 3 \\ 4 & 2 & 2 & 3 & 1 & 4 & 7 & 8 \\ 1 & 7 & 8 & 2 & 0 & 2 & 6 & 2 \\ 0 & 5 & 4 & 7 & 9 & 6 & 4 & 4 \\ 6 & 3 & 1 & 0 & 7 & 8 & 3 & 6 \end{bmatrix} \tag{3.31}$$

Convolutional layers act as feature detectors on the provided input. In order for this to happen, the initial data are filtered and the convolved matrix D_c with dimensions $[m,n]$ is generated. The filtering is achieved with the use of a filter kernel f which is also a matrix with $[u,h]$ dimensions. The filter, in a reading-like sequence, starts from the top left corner of D and moves to the right with a step equal to s. In each step, a value of D_c is calculated. Once the filter has been applied to the outer right position of D, the filter moves to the left edge under the starting edge. This procedure continues until all D has been filtered. It can be seen that the dimensions of D_c can be user-defined as they depend on the size of s and on they way

values on the edges of D are treated. The dimensions of D_c can be calculated as follows:

$$m = \frac{x-u}{s} + 1$$
$$n = \frac{y-h}{s} + 1 \tag{3.32}$$

In case the user wishes for D and D_c to have identical dimensions, three approaches can be applied with each one having different affect on the proposed D_c. According to the first one, dimensions of D are increased, by adding rows and columns on all edges of D, while the added data value is equal to zero. With the use of this method, the importance of features possibly existing in the boundaries of D is reduced. When this methodology is used, the size of D_c is calculated as follows:

$$m = \frac{x-u+2z}{s} + 1$$
$$n = \frac{y-h+2z}{s} + 1 \tag{3.33}$$

where z is the number of zero-filled rows that are added in one of the edges. The second method proposes to increase the dimensions of D_c by adding rows and columns on the ages but the values of the added data is equal to the nearest calculated value. The third methodology proposed to repeat the outer values of D outside the edges of D_c as many times as needed. With this method, it is possible to increase importance of features in outer areas of data which may not be significant. The mathematical expression of convolution via filtering can be expressed as follows [63]:

$$D_c(i,j) = (D * f)(i,j) \tag{3.34}$$

For each separate filter f that is applied to D, an extra feature map is created. The hyper-parameters of a CNN as presented are:

1. the number of filters to be used
2. the size of the filter
3. the step of the filter (stride)
4. the method used for controlling the size of the feature map D_c

Usually, the convolution layer is used along with an activation function. The typical activation functions used are:

1. Sigmoid function $f(x) = s(x) = \frac{1}{1+e^{-x}}$

2. Hyperbolic tangent function $f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$

3. Rectified linear unit $f(x) = max(0,x)$

Rectified linear unit is actually the most commonly used activation function as it offers fast feature detection.

The pooling layer usually follows the convolution layer and is used for reducing the size of input. This is performed as not all information existing in D is important for the network. Additionally, from the perspective of computational cost, it is preferable to perform calculations on small 3D matrices than large 2D ones. For the above reasons, it is preferable to reduce the width and height of data handled by the CNN while increasing depth is also preferable.

The functionality of the pooling layer is similar to the convolution layer. For the values inside a pooling area P of dimensions [p,p], which moves the way the convolution filter does, pooling is performed. Supposing that a convolved matrix D_c of size $[m,n]$ is undergoing pooling where the pooling area P has dimensions $[p,p]$ and the movement step of P is equal to s_p, the size of the pooled feature map D_p is equal to $[k,l]$ where k and l are calculated as follows:

$$\begin{aligned} k &= \frac{m-p}{s_p} + 1 \\ l &= \frac{n-p}{s_p} + 1 \end{aligned} \tag{3.35}$$

There are three types of pooling usually used [63]:

1. Max pooling, where the max of all values in the pooled area is assigned to the corresponding position of D_p

2. Average pooling, where the mean of all values in the pooled area is assigned to the corresponding position of D_p

3. $L^2 - norm$ pooling, where the $L^2 - norm$ of all values in the pooled area is assigned to the corresponding position of D_p

Among these types, in modern architectures max pooling is mainly preferred.

Fully connected layers resemble typical neural networks where all nodes of one layer are connected to all nodes of the next layer and an activation function is used according to desired output data and functionality of the CNN (classification, regression, etc.). In some CNN architectures, normalization layers are used for to increase training efficiency by reducing over-fitting. CNNs are trained with the use of back-propagation algorithm, similar to other neural networks as due to network's grouped convolution and pooling along with the reduced

size of fully connected layers, the total population of parameters to be trained is significantly small with respect to network size.

The previously described DL architectures (DBNs and CNNs) are used in Chapters 4, 5 and 6 with significant success in accelerating topology optimization application and as a generative design method.

Chapter 4

Applied Deep Learning on Topology Optimization

This chapter focuses on exploring new methods for obtaining fast and computationally "cheap" topology optimization (TO) results. It is known that all approaches available for solving TO problems are significantly computationally heavy even for problem parameters which are on a research level and not on the detail level necessary in construction. Though computer hardware available are constantly improving, either in terms of Central Processing Unit (CPU) and/or Graphics Processing Unit (GPU) with reference to their speed and memory per core and number of cores, the size and complexity of TO problems remain time and computational load greedy. As this drawback concerns practically everyone who works in the field of TO, several approaches have been proposed for reducing the computational load of TO mainly via incorporating parallel programming in CPU or GPU [16, 24, 45, 138]. In this part of the thesis, the possibility of developing new methods for TO, focused on accelerating the TO procedure by dramatically reducing its computational cost is examined. The implemented research focused of taking advantage of the capabilities of up-to-date deep learning methods in combination with established TO methodologies. In detail, DL-TOP method is proposed, which focuses on predicting a final output of TO based on premature TO results.

4.1 Topology Optimization

Topology optimization focuses on discovering the optimal distribution of material inside a predefined domain and under predefined loading and supporting conditions with respect to structural performance [194] by integrating mathematics and computational calculus methods.

As interest in TO has risen drastically, applications of TO can be found in several fields like aerospace design [230], implants manufacturing [216], architectural design [35], material design [192], fluid mechanics [153], structural design [101] and more.

Research interest in TO began to rise after the work presented on the homogenization method [13]. Since then, several other approaches have been proposed with the basic ones being [194]:

1. Density method
2. Level-Set method
3. Topological derivative method
4. Phase field method
5. Evolutionary method

Density method and its most well known formulation, Solid Isotropic Material with Penalization (SIMP) that was firstly introduced around 1990 [12, 229, 147], is using power-law for simplifying the homogenization method. TO with density method in structural problems can be summarized in the following expression:

$$\begin{aligned} &Minimize \quad F(x) \\ &with \quad respect \quad to: \\ &\qquad K * U = F \\ &\qquad g(x) \leq 0 \\ &\qquad 0 \leq x \leq 1 \end{aligned} \tag{4.1}$$

where $F(x)$ is the objective function, usually equal to the compliance of the system, x is the density variable vector, K is the global stiffness matrix, F and U are the loading and displacement vectors respectively and $g(x)$ are the problem constraints (volume fraction, etc.). A more thorough presentation of SIMP can be viewed in a following section.
The Level-Set method [3, 215], is based on the principle that the contour of a scalar function $\phi(x)$, named level-set function, defines the optimized design's boundaries when set equal to zero while the optimized domain is defined by areas where the $\phi(x)$ is positive. In an iterative manner, $\phi(x)$ is updated through the Hamilton-Jacobi equation as follows:

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| * V \tag{4.2}$$

where V, known as speed function, is used for defining the movement step of the contour. Topological derivative method was firstly introduced by the bubble method [47]. As a method it is based on defining the affect of inputing an elemental hole inside a domain. Information acquired, is used for making decision on where holes will be placed.
The phase field approach for topology optimization [17], originating from the natural phenomenon of solid-liquid transition [29], is based on applying a phase field function ϕ on the initial domain and separating it into two phases. This separation is based on a constant variation of the density variables [38].
Evolutionary method uses heuristic information for alternating the presence or absence of material in a discretized domain in a binary manner. The most well known method of this genre is Evolutionary Structural Optimization (ESO), proposed in 1993 [221] while many modified versions have been proposed [169, 168]. In an attempt to formulate TO problems in an mathematical manner, the following data need to be predefined:

1. Initial domain Ω to be optimized
2. Desired volume fraction of the optimized domain with respect to the initial one v_t
3. Boundary conditions Γ and
4. Loading conditions P

The boundary conditions Γ are defined as the total of sub-parts Γ_0, Γ_p, Γ_s, Γ_u as follows:

$$\Gamma = \Gamma_u \cup \Gamma_0 \cup \Gamma_p \cup \Gamma_s \tag{4.3}$$

where Γ_u are the support conditions, Γ_0 are the geometric limits of the initial domain Ω, Γ_p is the area of Ω where the external loads are applied and Γ_s is the non-optimizable area. The above formulation can be seen in Fig. 4.1.

4.1.1 Solid Isotropic Material with Penalization - SIMP method

Solid Isotropic Material with Penalization (SIMP) is one of the most established approaches in structural topology optimization (STO). As previously stated, the goal of STO is the optimal distribution of material inside a certain domain under defined loading and support conditions with respect to performance. The most commonly used performance indicator is the compliance C of the structure. As the domain Ω is discretized into n finite elements, the distribution of material is expressed by the x_i density values where $i \in [1,\ldots,n]$ and $x_i \in (0,1]$ with $x_i = 0$ indicating that no material is present on the i_{th} finite element and

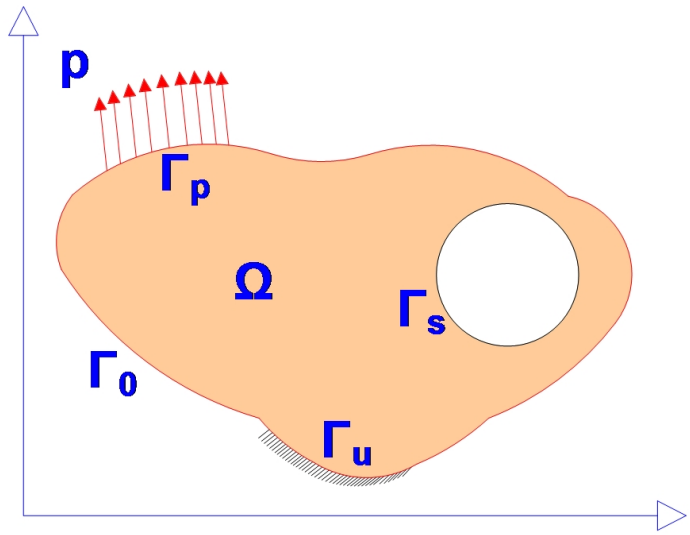


Fig. 4.1 Topology optimization formulation.

$x_i = 1$ indicating that the i_{th} finite element is filled with material. As a result of the above, Eq. 4.1 is transformed as follows:

$$\begin{aligned} &Minimize \quad C(x) = F^T * U(x) \\ &\quad with \quad respect \quad to: \\ &\qquad\qquad K(x) * U(x) = F \\ &\qquad\qquad \frac{V(x)}{V_0} = V_t \\ &\qquad\qquad 0 < x \leq 1 \end{aligned} \tag{4.4}$$

where $C(x)$ is the system's compliance for a given density vector x, $K(x)$ is the global stiffness matrix F and $U(x)$ are the loading conditions vector and the global displacements vector and $V(x), V_0, V_t$ are the volumes corresponding to density vector x, the initial volume for $x = x_0$ and the targeted volume of the optimized domain. In SIMP approach, the Young modulus E is correlated via power law to the density value of each finite element as follows:

$$E_x(x_i) = x_i^p E^0 \iff K_x(x_i) = x_i^p K^0 \tag{4.5}$$

where p is a penalization parameter usually setting $p = 3$. The above correlation is used for pushing SIMP towards generating density values x_i close to the lower and upper bound of x

[115]. In Eq. 4.4, the compliance can be calculated as follows:

$$C(x) = F^T * U(x) \iff C(x) = U^T(x) * K(x) * U(x) \iff C(x) = \sum_{i=1}^{n} x_i^p U_i^T K_i^0 U_i \quad (4.6)$$

Accordingly, Eq. 4.4 can be written as follows:

$$\begin{aligned} &Minimize \quad C(x) = \sum_{i=1}^{n} x_i^p U_i^T K_i^0 U_i \\ &with \quad respect \quad to: \\ &\qquad K(x) * U(x) = F \\ &\qquad \frac{V(x)}{V_0} = V_t \\ &\qquad 0 < x \leq 1 \end{aligned} \quad (4.7)$$

In literature, the optimization problem described in Eq. 4.7 is handled either by MMA or OC algorithms that where described in Chapter 2.

4.2 DL-TOP - Deep Learning Accelerated Topology Optimization

Work on this part of the dissertation is targeted at enhancing computational efficiency of SIMP approach when applied to STO problems. As such problems are extremely heavy in computational demands, deep neural networks are exploited for accelerating the optimization procedure. The capability of Deep Belief Networks (DBNs) in discovering multiple representational levels of nonlinearity in data in pattern recognition problems, triggered the development of the methodology proposed, DL-TOP, based on DBNs and SIMP. More specifically, a DBN is calibrated on transforming the input data containing density fluctuation pattern of the finite element (FE) discretization provided by the initial steps of the SIMP approach to a new higher-level representation. This representation corresponds to the final density values distribution over the domain as obtained by SIMP. DL-TOP results are validated over several benchmark topology optimization test examples where a reduction of iterations larger than one order of magnitude with respect to the ones that were originally required by SIMP is achieved. The gain through DL-TOP is analogous to the size of the TO problem.

4.2.1 DL-TOP methodology description

DL-TOP is a specially tailored methodology for accelerating SIMP approach in STO problems. For this reason, the DBN is used for predicting a close-to-final density value for each FE of the initial domain, according to initial densities produced in SIMP early iterations. In order for the prediction to be accurate, the DBN is trained once on a typical topology optimization problem before being applied to any STO problem regardless differences in mesh and domain dimensions, mesh type, loading conditions, desired final density, filter value, etc.

Assuming, without loss of generality, that a structured FE mesh discretization of ne_x, ne_y, ne_z FEs per axis is implemented on a rectangular domain. The population of created FEs ne given by $ne = ne_x * ne_y * ne_z$. According to SIMP, a density value d_i is assigned to each ne_i FE as an initialization step while d_i is updated in every iteration of SIMP approach. The fluctuation of density value d_i of the i_{th} FE with respect to the iteration step t can be expressed by a function with respect iteration t:

$$d_i = F(t) \qquad \forall\, i \in [1, ne] \tag{4.8}$$

The fluctuation of density per SIMP iteration for several FEs can be seen in Fig. 4.2. It is more than obvious that fluctuation varies drastically for different FEs due to their position in the domain with respect to position of loads, support positions, etc. Each FE presents a different optimization history of density values per SIMP iteration corresponding to a sequence of discrete time-data similar to a time-series. Density initialization for each FE is 0.40 which is derived from the targeted volume constraint which is equal to 40% of the initial volume. The above uniform starting density value for all FEs with respect to the volume fraction value constraint is a common practice in SIMP implementation [193, 4]. SIMP computational load depends on the population of FEs resulting to significant loads even for non-densely discretized domains. This problem is magnified in finer 3D meshes. As an example, addressing the STO problem of a 3D bridge test case consisting of 83,000 FEs can take up to 7 hours for performing 200 iterations of SIMP in serial CPU execution while the same problem executed in parallel GPU environment requires 1 hour [101] even for such a small number of FEs in the example. DL-TOP methodology can be applied to both serial or parallel, CPU or GPU execution implementations.

DL-TOP methodology can be described as a two-phase procedure. In the first step, SIMP performs a small number of initial iterations which are used as input data for the DBN. The DBN, based on the input, proposes an optimized domain at the end of the first phase. In the second phase, SIMP performs fine-tuning on the DBN-proposed optimized domain. A

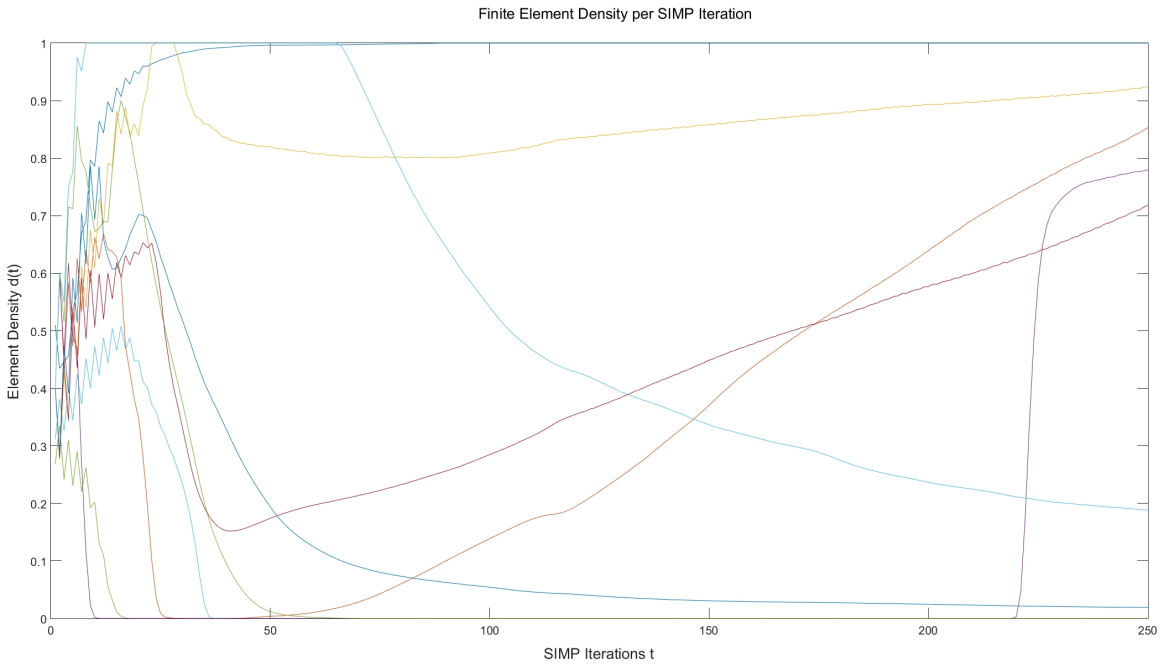


Fig. 4.2 Fluctuation of density of various finite elements with respect to the SIMP iterations

population of thirty six iterations of SIMP are executed for creating the necessary DBN input vectors of density per iteration per FE. Once the input is evaluated by the DBN, a discrete jump from the thirty sixth iteration to a close to final density per FE is made by the DBN. The methodology completes with the fine-tuning performed by SIMP. A flowchart of DL-TOP methodology is presented in Fig. 4.3, while the application of the two-phase methodology in the case of a single finite element is shown in Fig. 4.4. The abscissa of Figure 5(b) represents the iterations performed by SIMP while the ordinate corresponds to the density value of the single finite element.

The key feature of DL-TOP is that after training on a simple example, it can be successfully applied to different cases of STO without need for retraining. This is achieved by the fact that each finite element is handled separately without no information needed on its position in the domain, loading and boundary conditions of the domain, etc.

Classification problems are a challenging area of predictive modeling. Contrary to regression predictive modeling, classification models require information also on the complexity of a sequence dependence among the input parameters. In the case of STO the early density values represent the sequence dependence information that needs to be provided as inputs to the proposed (classification) methodology. The sequence of discrete-time data, i.e. the density value for every FE and the T iterations are generated by SIMP approach and stored

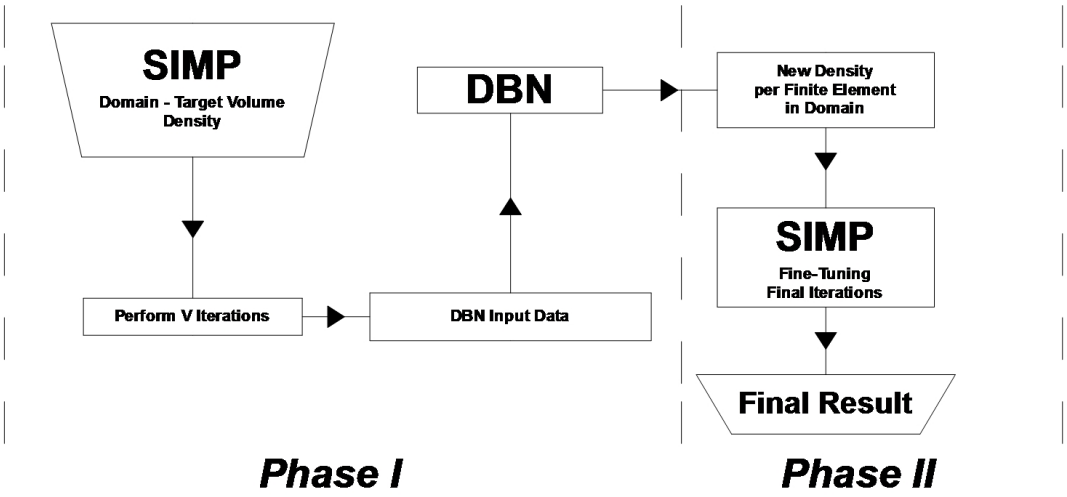


Fig. 4.3 DL-TOP methodology flowchart

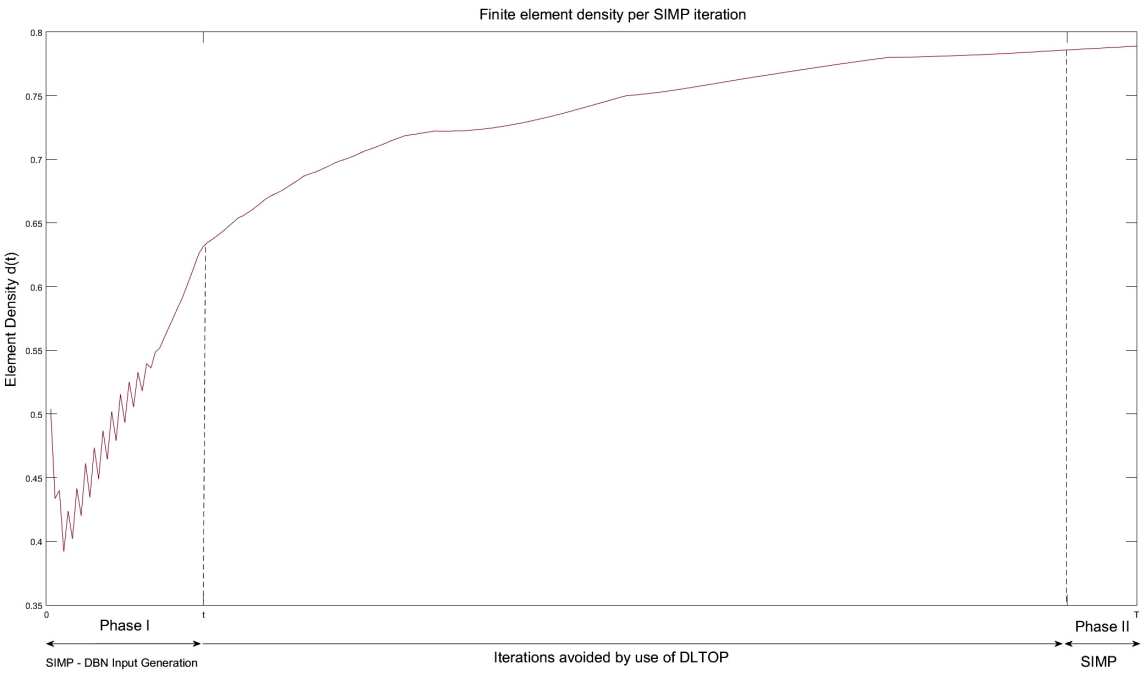


Fig. 4.4 DLTOP methodology implementation for a single FE

in matrix D presented below:

$$D = \begin{bmatrix} d_{1,1} & d_{1,2} & \dots & d_{1,T} \\ d_{2,1} & d_{2,2} & \dots & d_{2,T} \\ \vdots & \vdots & \ddots & \vdots \\ d_{ne,1} & d_{ne,2} & \dots & d_{ne,T} \end{bmatrix} \tag{4.9}$$

where T denotes the maximum iterations needed by SIMP to achieve convergence. A small part of the optimization procedure equal to the first t iterations is used as time-series input data for training the DBN while the vector of the final iteration of SIMP approach corresponding to the T_{th} column of density matrix D is used as the target vector of DBN training.

$$\begin{array}{c} \left[\begin{array}{cccc|ccc|c} d_{1,1} & d_{1,2} & \dots & d_{1,t} & d_{1,t+1} & \dots & d_{1,T-1} & d_{1,T} \\ d_{2,1} & d_{2,2} & \dots & d_{2,t} & d_{2,t+1} & \dots & d_{2,T-1} & d_{2,T} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{ne-1,1} & d_{ne-1,2} & \dots & d_{ne-1,t} & d_{ne-1,t+1} & \dots & d_{ne-1,T-1} & d_{ne-1,T} \\ d_{ne,1} & d_{ne,2} & \dots & d_{ne,t} & d_{ne,t+1} & \dots & d_{ne,T-1} & d_{ne,T} \end{array}\right] \\ \begin{array}{ccc} \textit{Training Sample} & \textit{Not Used} & \textit{Target} \end{array} \end{array} \tag{4.10}$$

4.2.2 Training dataset

Classification of certain input data based on an ML-based calculated output is one of the tasks that DL algorithms show remarkable results. Training on multiple datasets is a prerequisite for a successful classification execution of DL methods. Data available to the user are separated into three smaller datasets, each one used in a different phase of the DL application task. Initially, the DL model is calibrated on a series of data named training dataset; while successively, the calibrated model (metamodel) is used for generating outputs based on different input data, named as validation data. In the final step of this procedure, the test dataset is used for calculating an unbiased assessment of the final model. The construction of the training dataset of DL-TOP methodology is described in the following section. DL-TOP performs classification of data consisting of density of FE per SIMP iteration. This is based on discovering higher order features in the patterns of initial density fluctuations of each FE and matching these patterns with the several density final values. In this direction, the training/validations/testing datasets are composed of sequences of the density values derived from the implementation of SIMP approach. As previously stated, the performance of DL-TOP methodology is independent of the test example used for developing the training dataset. In order to prove this claim, two distinct training datasets are created and compared with each other.

The two databases are constructed by two basic and simple 2D benchmark test examples of topology optimization, the simply supported beam and the cantilever beam, respectively. The two test exampled considered for constructing the training datasets are shown in Fig. 4.5 (simply supported beam Fig. 4.5(a) and 4.5(b); cantilever beam Fig. 4.5(c) and 4.5(d)). In order to derive a well-constructed and diversified training dataset, two height to length

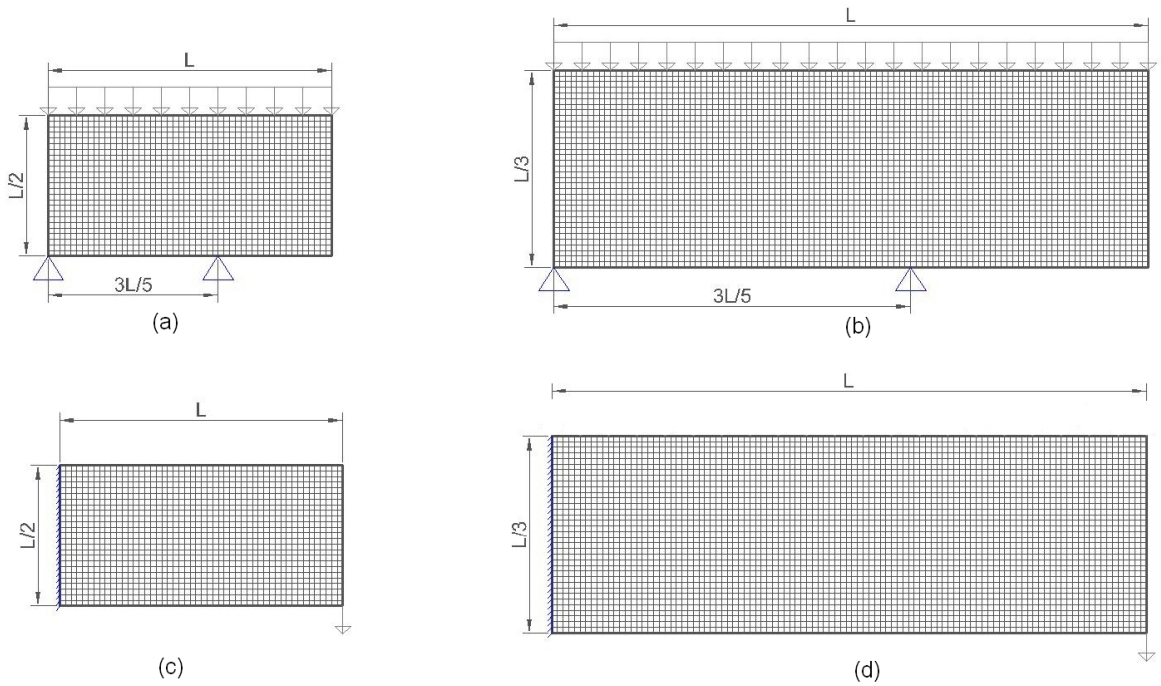


Fig. 4.5 Training datasets generation, indicative FE discretization for: (a) 1:2 and (b) 1:3 cantilever beam; (c) 1:2 and (d) 1:3 simply supported beam.

ratios are implemented (i.e. 1:2 and 1:3) for each test example while eight different FE mesh discretizations were adopted from sparse to dense meshing. It is worth mentioning that while increasing mesh density, the size of the unit FE remains the same, thus finer discretization results into larger domain sizes). Specifically, in order to build the datasets the fol-lowing discretizations were used $D = \begin{bmatrix} D_K & D_{3K} & D_{6K} & D_{10K} & D_{20K} & D_{40K} & D_{60K} & D_{100K} \end{bmatrix}^T$ where D_K stands for the samples generated for a FE mesh discretization of the order of 1,000 FEs while D_{10K} stands for meshes consisting of 10,000 FEs. Therefore, the samples used to derive the two training datasets are composed by the iteration histories of all the FEs when

SIMP is applied to the following STO problems:

$$
\begin{aligned}
&D_K \begin{cases} 1:2 \begin{cases} ne_x = 50 \\ ne_y = 25 \end{cases} \\ 1:3 \begin{cases} ne_x = 60 \\ ne_y = 20 \end{cases} \end{cases} \\
&D_{3K} \begin{cases} 1:2 \begin{cases} ne_x = 80 \\ ne_y = 40 \end{cases} \\ 1:3 \begin{cases} ne_x = 105 \\ ne_y = 35 \end{cases} \end{cases} \\
&D_{6K} \begin{cases} 1:2 \begin{cases} ne_x = 110 \\ ne_y = 55 \end{cases} \\ 1:3 \begin{cases} ne_x = 135 \\ ne_y = 45 \end{cases} \end{cases}
\end{aligned}
\qquad
\begin{aligned}
&D_{10K} \begin{cases} 1:2 \begin{cases} ne_x = 145 \\ ne_y = 73 \end{cases} \\ 1:3 \begin{cases} ne_x = 180 \\ ne_y = 60 \end{cases} \end{cases} \\
&D_{20K} \begin{cases} 1:2 \begin{cases} ne_x = 200 \\ ne_y = 100 \end{cases} \\ 1:3 \begin{cases} ne_x = 245 \\ ne_y = 82 \end{cases} \end{cases} \\
&D_{40K} \begin{cases} 1:2 \begin{cases} ne_x = 285 \\ ne_y = 143 \end{cases} \\ 1:3 \begin{cases} ne_x = 350 \\ ne_y = 117 \end{cases} \end{cases}
\end{aligned}
\qquad
\begin{aligned}
&D_{60K} \begin{cases} 1:2 \begin{cases} ne_x = 350 \\ ne_y = 175 \end{cases} \\ 1:3 \begin{cases} ne_x = 425 \\ ne_y = 142 \end{cases} \end{cases} \\
&D_{100K} \begin{cases} 1:2 \begin{cases} ne_x = 450 \\ ne_y = 225 \end{cases} \\ 1:3 \begin{cases} ne_x = 550 \\ ne_y = 184 \end{cases} \end{cases}
\end{aligned}
\tag{4.11}
$$

Loading conditions of the two examples used are presented in Fig. 4.5. The desired volume fraction of the final domain is set equal to 40% of the original one in both examples and a predefined value of density filter is also used. In the first training test example the radius chosen is equal to two elements for all discretizations and for the second one the radius ranges from three elements in the case of the 1,000 elements discretization to fifteen elements in the 10,000 elements one in order to diversify the database construction method. All results of the simply supported beam are joined for formulating the first dataset while all results of the cantilever beam formulate the second dataset. Each dataset contains almost 480,000 density sequences. The population of initial SIMP iterations t was chosen to be equal to 36 based on the 15% of the weighted sum of SIMP iterations needed for convergence with reference to the FE discretization. Database construction in ML applications is defined the principle that all patterns and classes existing must be represented equally in the database. Since DL-TOP is tailored made for TO problems, a different approach is also examined in this dissertation. According to that, all classes are represented not equally but according to the frequency of their appearance. In our case, although the values regarding the final density of finite elements range in (0, 1], comparing the two datasets it can be seen that they represent two completely different distributions of the density values in this range. The simply supported beam training dataset mainly consists of values equal to either zero or one (according to the frequency of their appearance on the final result) while the cantilever beam training dataset consists of many varying values in the range of [0.1, 0.9], similarly to equal representation of all classes principle. This difference on the analogies of the classes explains

the difference in terms of performance of the two datasets. In an effort to thoroughly examine the performance of the proposed methodology, it is important to assess both datasets as the outcome of the classification generated from each one is expected to have differences.

4.2.3 DBN calibration

As in all ML models, DL-TOP is developed according to the calibration and implementation phases. The calibration phase is performed once for each dataset presented. The DBN used in DL-TOP methodology identifies the patterns of density fluctuation for each FE with respect to the final density of the FE as proposed by SIMP. Prior to the training procedure, SIMP is applied for creating the training samples i.e. solve specific topology optimization problems as denoted in Eq. 4.11. Then, the training dataset is formed, composed by the inputs (i.e. *ne* vectors consisted of the FEs' densities for the first t iterations) and the target outputs (i.e. a vector of size *ne* corresponding to the densities achieved at the T_{th} final iteration of SIMP). These training samples are then used for calibrating the DBN as described in the following expression:

$$\begin{array}{ccccccc} [d_{1,1} & d_{1,2} & \dots & d_{1,t}] & & \rightarrow & d_{1,T} \\ [d_{2,1} & d_{2,2} & \dots & d_{2,t}] & & \rightarrow & d_{2,T} \\ & \vdots & & \vdots & \ddots & & \vdots \\ [d_{ne-1,1} & d_{ne-1,2} & \dots & d_{ne-1,t}] & & \rightarrow & d_{ne-1,T} \\ [d_{ne,1} & d_{ne,2} & \dots & d_{ne,t}] & & \rightarrow & d_{ne,T} \\ & Input & & & & & Target \end{array} \tag{4.12}$$

Thus, during the calibration phase DBN is adjusted for generating the non-linear transformations for each FE from the density fluctuation pattern of the first t iterations to the final density $d_{i,T}^{DBN}$, $i = 1, 2, \dots, ne$. The DBN output values represent a classification of each FE of the design domain with respect to the $d_{i,T}^{DBN}$ density values.

4.2.4 DL-TOP implementation

Once the calibration phase is completed, the DBN metamodel can be used alongside SIMP in the DL-TOP implementation phase for accelerating the TO procedure according to the following expression for every test example:

$$DBN([d]_{1:ne,1:t}) \Rightarrow [d]_{1:ne,T} \quad \forall\ [d]_{1:ne,1:t}, ne, t, T \tag{4.13}$$

The implementation phase of the DL-TOP methodology is composed by three steps:

- SIMP generates DBN input
- DBN predicts final volume class according to input
- SIMP fine-tunes DBN proposed result

In the first step, SIMP is used for generating the sequence of t samples for each FE in the design domain mesh. In the second step, the calibrated DBN is applied for deriving the $d_{i,T}^{DBN}$ density value of each FE of the domain. Worth mentioning that the computing requirements for applying DBN are small (in the order of 1.7E-04 seconds on an i7-3610QM processor), but in the case of million or even billion of finite elements it might become significant. However, the application of DBN metamodel is independent for each finite element and therefore can be performed in parallel without any interprocess communication. In the final part of DL-TOP, SIMP is fed with the $d_{i,T}^{DBN}$ density values of the FEs of the design domain. This step is necessary for correcting any defects present in DBN output. A flowchart of the DL-TOP methodology is presented in Fig. 4.3 while the pseudocode is presented in Fig. 4.6. According to the pseudocode of Fig. 4.6, lines #3 to #5 represent the DBN input generation

```
1  Initialization
2      # Run SIMP for t iterations
3        For i := 1 To t Do Begin
4            Calculate d_i^i        # Calculate Initial Density Data by SIMP
5        End
6      # Feed t density data to DBN and extract close-to-final-value
7        For k := 1 To s Do Begin
8            Calculate d_k^T        # Calculate DBN-proposed optimized domain
9        End
10     # Use SIMP for fine-tuning
11       While Termination Criterion not satisfied Do Begin
12           Calculate d_m^T        # Fine-tune DBN-proposed optimized domain by SIMP
13       End
14 End
```

Fig. 4.6 Pseudo-code of DL-TOP methodology

by SIMP approach of Phase I. These input date are used to feed the DBN and the optimized domain to be derived. This is presented in lines #7 to #9. The density values of the optimized domain as proposed by the DBN are fine-tuned by SIMP approach in Phase II and the final DL-TOP proposed optimized domain is created (lines #11 to #13). The DBN used in this

study consists of three RBMs and the framework was developed based on work by Hinton and Salakhutdinov [76]. The number of training samples is equal to the total number of finite elements generated by the domains of the topology optimization problems of Eq. 4.11, i.e. *ne* input density vectors is equal to 480,000 while length t of each input vector was chosen equal to 36 recorded initial density calculations. The sizes of the three RBMs are equal to 30, 20 and 12. Two different classes are used, i.e. the output of the classification network is divided into 12 classes in the first one and in 3 classes in the second one as denoted below:

$$\begin{aligned}
&\text{Classification} \quad \text{Case} \quad \text{I} \\
&d_{i,T} = 0 \\
&d_{i,T} \in \begin{cases} (0,0.1] \Rightarrow d_{i,T} = 0.05 \\ (0.1,0.2] \Rightarrow d_{i,T} = 0.15 \\ (0.2,0.3] \Rightarrow d_{i,T} = 0.25 \\ (0.3,0.4] \Rightarrow d_{i,T} = 0.35 \\ (0.4,0.5] \Rightarrow d_{i,T} = 0.45 \\ (0.5,0.6] \Rightarrow d_{i,T} = 0.55 \\ (0.6,0.7] \Rightarrow d_{i,T} = 0.65 \\ (0.7,0.8] \Rightarrow d_{i,T} = 0.75 \\ (0.8,0.9] \Rightarrow d_{i,T} = 0.85 \\ (0.9,1) \Rightarrow d_{i,T} = 0.95 \end{cases} \\
&d_{i,T} = 1
\end{aligned} \tag{4.14}$$

$$\begin{aligned}
&\text{Classification} \quad \text{Case} \quad \text{II} \\
&d_{i,T} \in \begin{cases} [0,0.4] \Rightarrow d_{i,T} = 0 \\ (0.4,0.7] \Rightarrow d_{i,T} = 0.50 \\ (0.7,1] \Rightarrow d_{i,T} = 1 \end{cases}
\end{aligned} \tag{4.15}$$

4.2.5 Test examples

Testing of DL-TOP methodology performance is divided into two parts. At first, a parametric investigation is performed with respect to different training datasets, RBM training parameters and size of output classes. The test examples used in this part are 2D benchmark test from literature. In the second part, the efficiency and robustness of DL-TOP is evaluated on five test examples (2D and 3D) aquired from literature. As for the SIMP approach, without loss of generality OC and MMA algorithms are chosen for solving the topology optimization problems at hand. The penalization factor is taken equal to 3 and is not changed for all test

examples. Filtering is implemented by taking into consideration the weighted derivative of the adjacent elements for the calculation of each elements derivative in both 2D and 3D examples. Worth mentioning that no thresholding was used in any of the test examples examined below (i.e. no thresholding has been applied to any of the optimized domains presented in this part of the thesis). The codes used for solving the TO problem are based on the 88-line code [4], its 3D variant and the PolyTop [206] code. It should be underlined that the filter type and radius remain the same in the cases examined in each test example described below, i.e. both during the implementation of the proposed methodology and the conventional SIMP approach. Thus, it is confirmed that the compliance values per example are obtained under exactly the same parametric conditions. The proposed DL-TOP methodology can easily be integrated with continuation of penalization techniques [58, 174, 124] as DL-TOP only deals with the volume time-history of each element regardless its position. The above-mentioned technics can be applied in SIMP Phases of DL-TOP without having to retrain the network.

Description of the five 2D benchmark test examples

Fig. 4.7 depict five 2D benchmark topology optimization test examples that are considered in order to perform the parametric investigation of DL-TOP while its performance is also evaluated. In the first example shown in Fig. 4.7(a), named as "short-beam (fine) test example", the discretization along the x and y axes is equal to $ne_x = 800$ and $ne_y = 150$, while the support conditions are four fixed joints placed at each corner of the domain. The single loading condition are two concentrated forces P along the y axis and applied in the middle of the span of the x dimension. The term "fine" is used because subsequently the same test example is studied using coarser FE mesh discretization. The second example shown in Fig. 4.7(b) is named as "antisymmetric test example". The discretization along the x and y axes is equal to $ne_x = 400$ and $ne_y = 400$, respectively. Support conditions refer to two fixed joints placed at the two right corners of the domain while the loading condition refer to two concentrated forces P along the x axis and applied in the middle of the span of the y dimension as seen in Fig. 4.7(b). The third one (Fig. 4.7(c)) is named as "column test example" and the discretization along the x and y axes is equal to $ne_x = 300$ and $ne_y = 500$, respectively. Support conditions refer to fully fixed boundary conditions along the x axis and starts at the $\frac{3}{8}ths$ of the x dimension. The loading conditions refer to four concentrated forces P along the y axis and applied with distance equal to $\frac{1}{3}rd$ of the x dimension. The fourth test example shown in Fig. 4.7(d) is named as "L-shape" and the discretization along the x and y axes are equal to $ne_x = 400$ and $ne_y = 400$, respectively. Support conditions refer to fully fixed boundary conditions along the x axis ending at the $\frac{1}{2}$ of the x dimension and the loading condition refer to a concentrated load P along the y

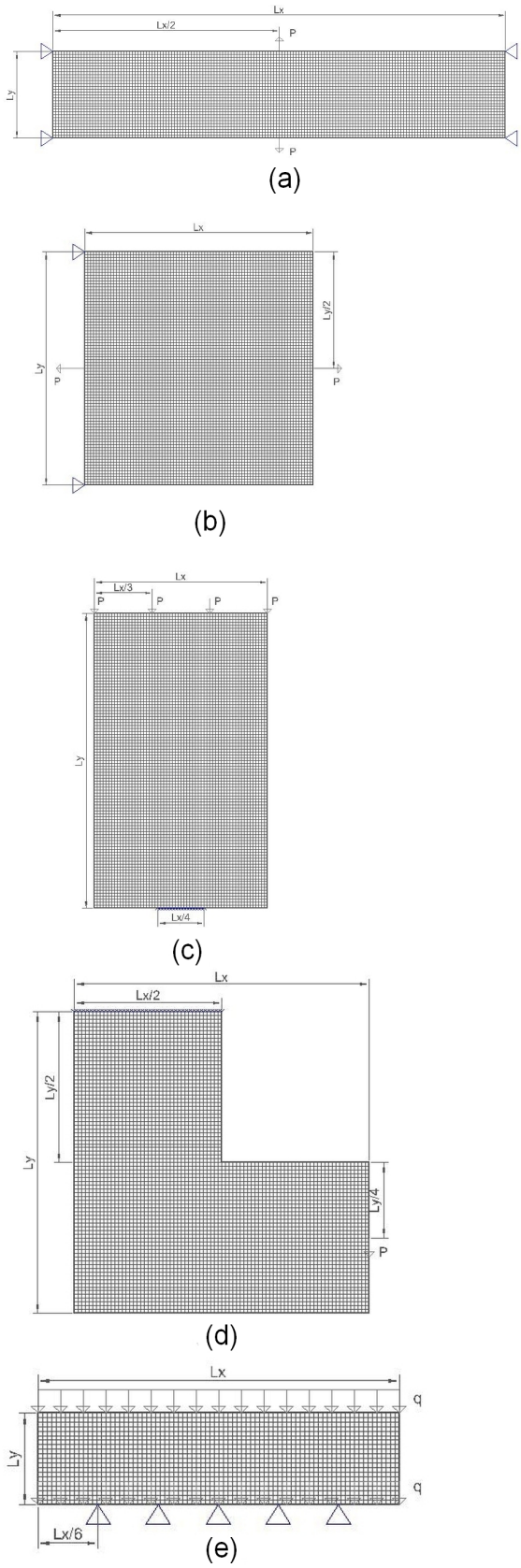


Fig. 4.7 2D test examples: (a) short-beam (fine), (b) antisymmetric, (c) column, (d) l-shape and (e) long-beam.

axis, applied in the $\frac{1}{4}th$ of the y dimension (Fig. 4.7). Finally, the fifth test example shown in Fig. 4.7(e) is named "long-beam test example" with the discretization along the x and y axis being equal to $ne_x = 400$ and $ne_y = 100$, respectively while the boundary conditions refer to five simple supports applied along the x axis with distance equal to $\frac{1}{6}th$ of the x dimension between them. The loading conditions refer to distributed force q along the y axis, applied on the top and bottom of the domain in the y dimension. In these 2D test examples, the volume fraction of the optimized domain is equal to 40% of the initial one and the filter radius is set equal to six elements for the short-beam (fine), antisymmetric, column and long-beam test examples while it is chosen equal to two elements for the L-shape test example. The standard sensitivity filter is implemented to all five test examples, while it should be stated that the parametric conditions remain the same for the original topology optimization and the proposed methodology. For each experiment, a record is kept regarding the total iterations needed and the objective function value when only SIMP is used and when acceleration by means of DBN is used.

Evaluation of parameters, training datasets and classification cases

The parametric investigation of DL-TOP methodology is performed with reference to:

- Two different datasets
- Ten different RBM training parameters sets
- Two different sizes of classes

The efficiency of the above factors' combinations is assessed on the five 2D benchmark topology optimization problems. As previously described, there are four parameters that define the learning behaviour of RBMs and in result the behaviour of DBNs. These parameters are: the learning rates of the weights e_w, the biases of the visible nodes e_b^v, of the hidden nodes e_b^h and of the weight cost w_c. The parameter values used in the RBM pretraining procedure are presented on Table 4.1 along with all the training characteristics. The ten different Parameter Sets (PS) defined in Table 1 are labelled as PS1 to PS10. In the back-propagation algorithm, 50 epochs were performed while in each epoch, 100 CG iterations were executed. The selected values were chosen accordingly to literature recommendations [74].

The results of DL-TOP methodology applied on the above test examples can be witnessed in Fig. 4.8 to 4.11 and Tables 2 to 5 where SB3, SB12, CB3 and CB12 stand for the simply Sup-ported Beam (SB) and Cantilever Beam (CB) datasets, respectively with 3 and 12 classes.

Table 4.1 RBM pre-training parameter value sets.

PS	e_w	e_b^v	e_b^h	w_c	Initial momentum	Epochs	Final momentum
1	1.00E-02	1.00E-02	1.00E-02	1.00E-04	0.5	500	0.90
2	5.00E-02	5.00E-02	5.00E-02	1.00E-04	0.5	500	0.90
3	1.00E-01	1.00E-01	1.00E-01	1.00E-04	0.5	500	0.90
4	1.00E-03	1.00E-03	1.00E-03	1.00E-04	0.5	500	0.90
5	3.00E-01	3.00E-01	3.00E-01	1.00E-04	0.5	500	0.90
6	1.00E-02	1.00E-02	1.00E-02	1.00E-03	0.5	500	0.90
7	1.00E-02	1.00E-02	1.00E-02	1.00E-02	0.5	500	0.90
8	1.00E-01	1.00E-01	1.00E-01	1.00E-05	0.5	500	0.90
9	6.00E-01	6.00E-01	6.00E-01	1.00E-04	0.5	500	0.90
10	1.00E-02	1.00E-02	1.00E-02	5.00E-05	0.5	500	0.90

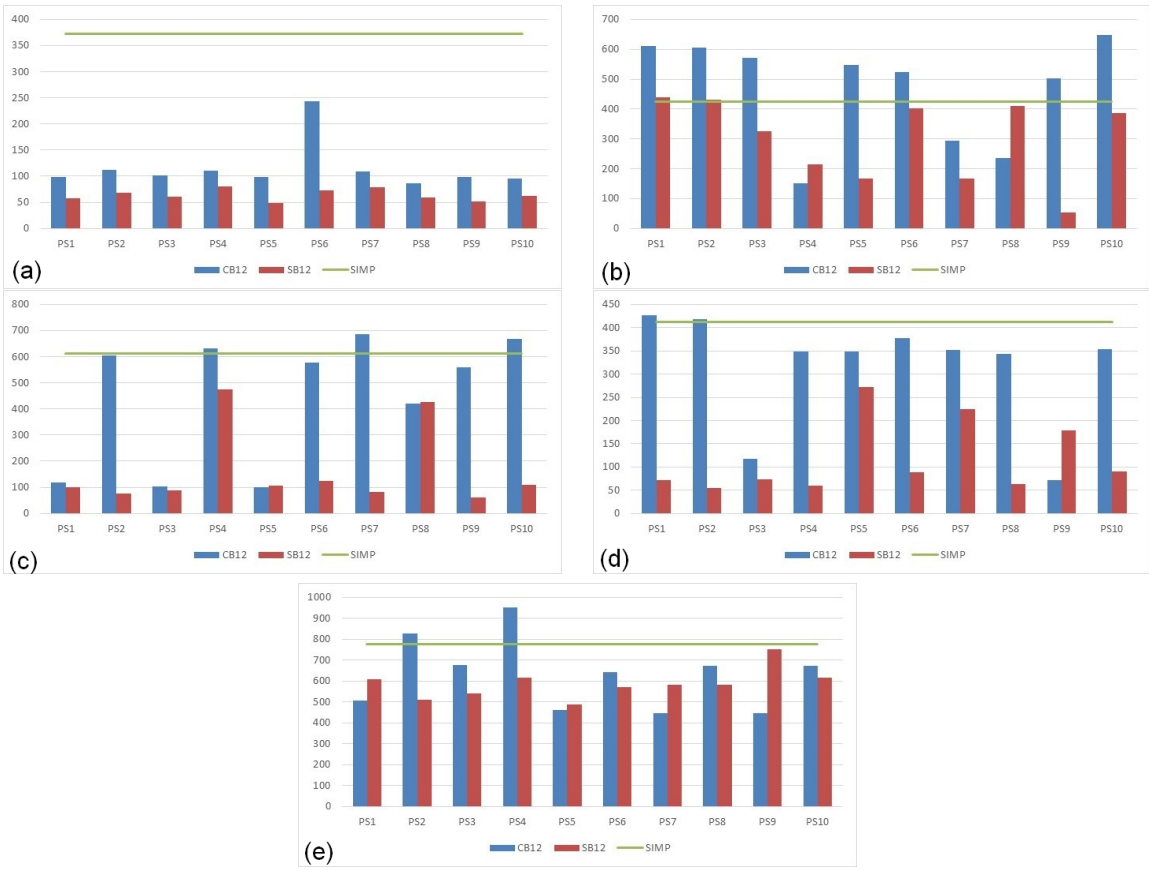


Fig. 4.8 Performance of classification twelve-Iterations: (a) short-beam (fine) test example, (b) antisymmetric test example, (c) column test example, (d) L-shape test example and (e) long-beam test example.

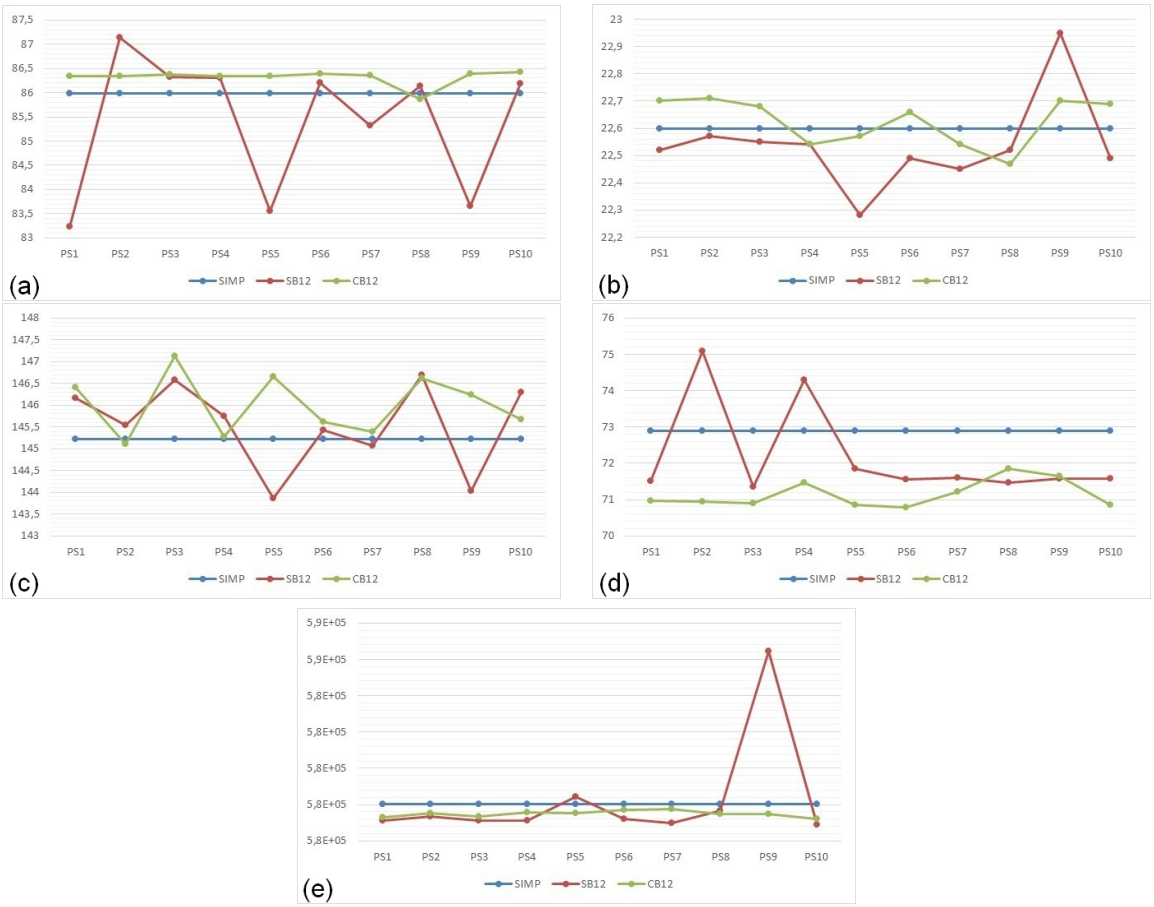


Fig. 4.9 Performance of classification twelve-Objective function value: (a) short-beam (fine) test example, (b) antisymmetric test example, (c) column test example, (d) L-shape test example and (e) long-beam test example.

Table 4.2 Average and variance of classification twelve performance-SB dataset.

Test Example	Iterations				Objective function value			
	SIMP	DLTOP			SIMP	DLTOP		DLTOP-SIMP Diff. (%)
		Average	Variation (%)	COV (%)		Average	COV (%)	
Short-beam (fine)	372	64	-82.82	16.22	85.99	85	1.55	-1.15
Antisymmetric	425	300	-29.48	43.58	22.60	23.00	0.70	1.77
Column	613	164	-73.18	87.92	145.22	146.00	0.64	0.54
L-shape	412	118	-71.41	62.94	72.89	72.00	1.76	-1.22
Long-beam	775	587	-24.27	11.77	577,015.85	577,237.00	0.48	0.04

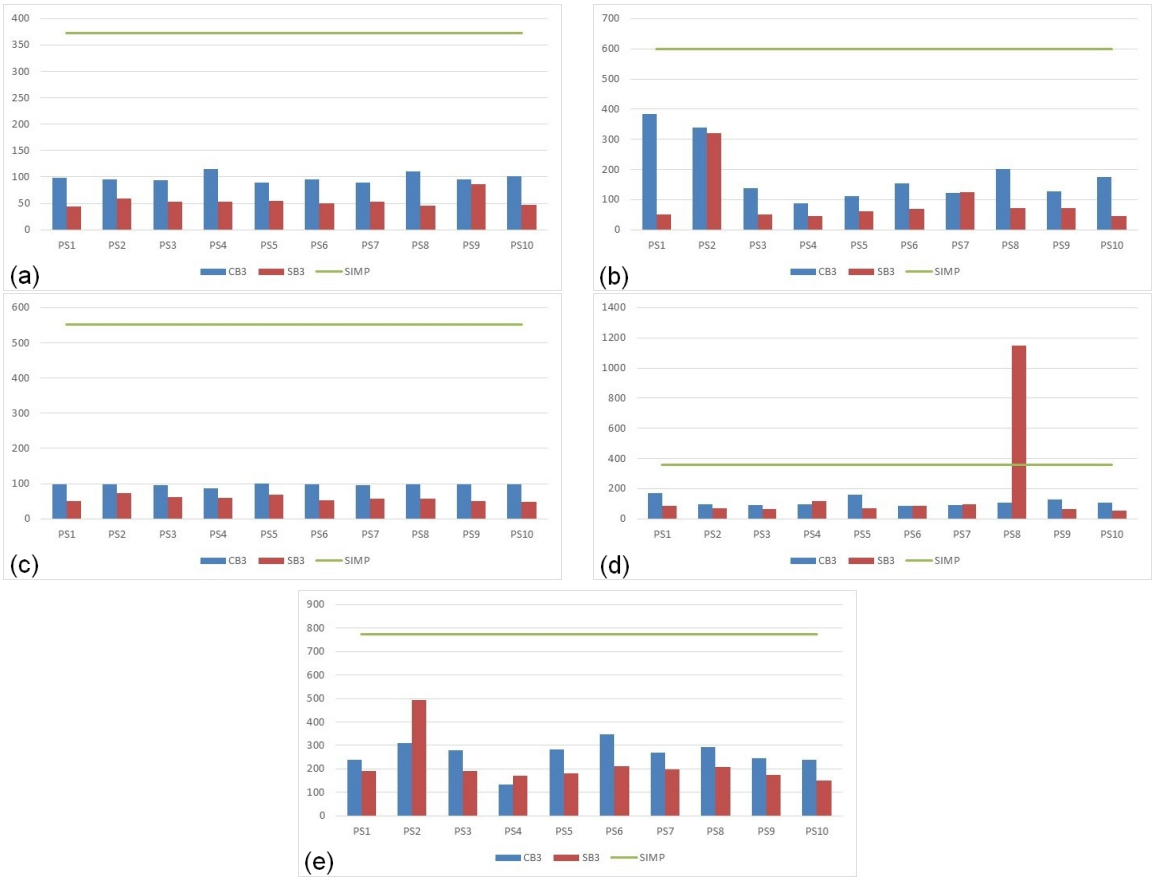


Fig. 4.10 Performance of classification three-Iterations: (a) short-beam (fine) test example, (b) antisymmetric test example, (c) column test example, (d) L-shape test example and (e) long-beam test example.

Table 4.3 Average and variance of classification twelve performance-CB dataset.

Test Example	Iterations				Objective function value			
	SIMP	DLTOP			SIMP	DLTOP		DLTOP-SIMP Diff. (%)
		Average	Variation (%)	COV (%)		Average	COV (%)	
Short-beam (fine)	372	115	-69.01	37.72	85.99	86.00	0.18	0.01
Antisymmetric	425	469	10.40	35.48	22.60	23.00	0.37	1.77
Column	613	446	-27.18	52.70	145.22	146.00	0.45	0.54
L-shape	412	31	-23.42	36.34	72.89	71.00	0.51	-2.59
Long-beam	775	630	-18.68	25.59	577,015.85	576,489.00	0.03	-0.09

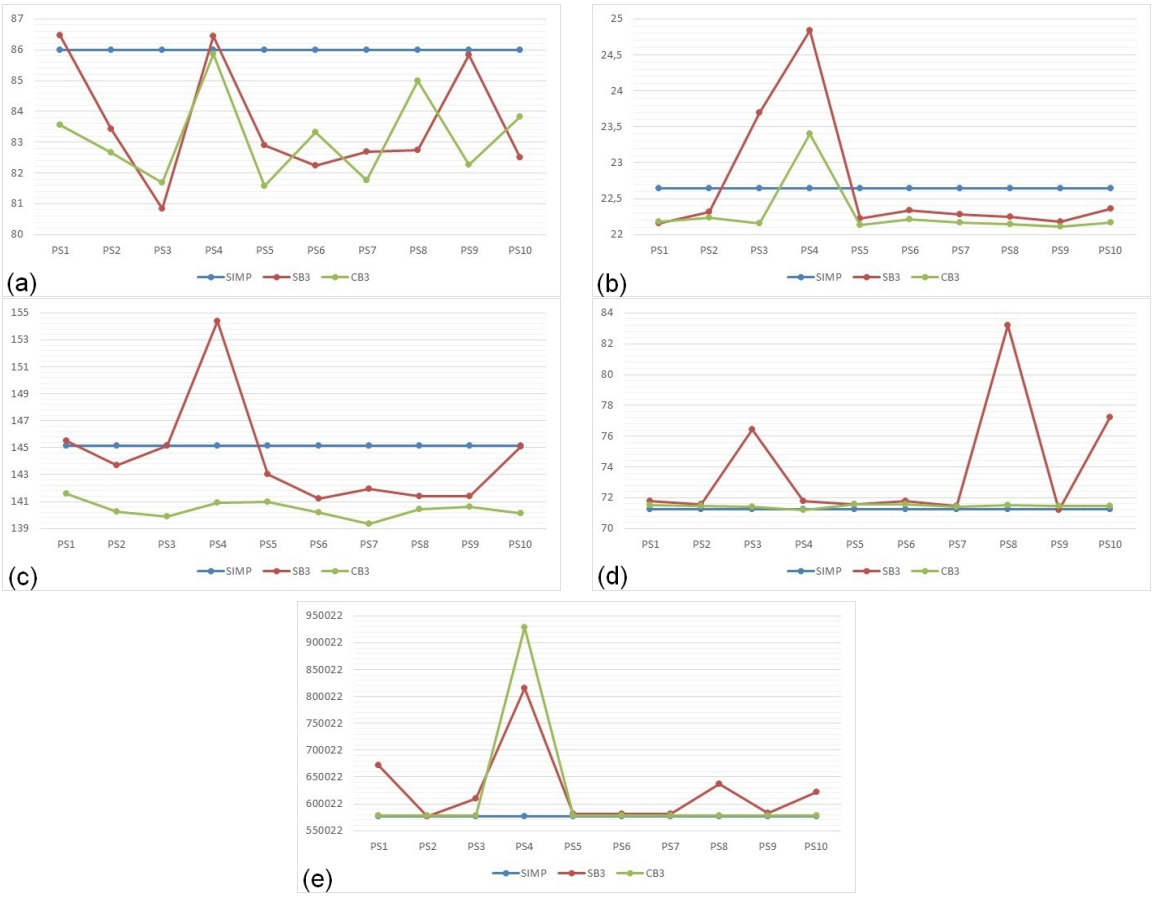


Fig. 4.11 Performance of classification three-Objective function value: (a) short-beam (fine) test example, (b) antisymmetric test example, (c) column test example, (d) L-shape test example and (e) long-beam test example.

Table 4.4 Average and variance of classification three performance-SB dataset.

Test Example	Iterations				Objective function value			
	SIMP	DLTOP			SIMP	DLTOP		DLTOP-SIMP Diff. (%)
		Average	Variation (%)	COV (%)		Average	COV (%)	
Short-beam (fine)	372	55	-85.35	20.75	85.99	84.00	2.21	-2.31
Antisymmetric	425	91	-78.56	87.48	22.60	23.00	3.72	1.77
Column	613	58	-90.57	13.08	145.22	144.00	2.58	-0.84
L-shape	412	186	-54.88	173.13	72.89	74.00	5.10	1.52
Long-beam	775	217	-71.96	43.17	577,015.85	625,716.00	11.13	8.44

Table 4.5 Average and variance of classification three performance-CB dataset.

Test Example	Iterations				Objective function value			
	SIMP	DLTOP			SIMP	DLTOP		DLTOP-SIMP Diff. (%)
		Average	Variation (%)	COV (%)		Average	COV (%)	
Short-beam (fine)	372	99	-73.49	7.94	85.99	83.00	1.66	-3.48
Antisymmetric	425	184	-56.73	51.27	22.60	22.00	1.67	-2.65
Column	613	96	-84.27	3.63	145.22	140.00	0.42	-3.59
L-shape	412	113	-72.52	24.73	72.89	71.00	0.15	-2.59
Long-beam	775	263	-66.04	20.65	577,015.85	612,929.00	17.15	6.22

Specifically, Fig. 4.8 and 4.10 depict the function evaluations required for DL-TOP methodology to converge for both databases, both classifications and PS sets. DL-TOP performance is also compared with SIMP alone with respect to necessary iterations for convergence (see the green line in Fig. 4.8 and 4.10). Fig. 4.9 and 4.9 illustrate the final objective function values achieved by DL-TOP, i.e. with reference to the two databases, two classifications and DBN training parameters; their performances are compared with those achieved by SIMP alone (see the green line in Fig. 4.9 and 4.11). The results concerning average objective function value and COV, shown in Tables 2 to 5, were calculated using the ten different sets of RBM training parameters (i.e. PS1 to PS10) as described previously. By studying the results presented in Fig. 4.8 and 4.10, it is obvious that the use of three output classes provides significantly better and more stable results than twelve classes, regardless of the training parameters of the DBN network. Additionally, the average computational efficiency performance of the SB dataset is better than the one of the CB dataset while in terms of objective function value, DL-TOP presents the same results regardless dataset used. As a general guide, it can be said that the simply supported beam represent the optimal choice for the database combined with three classes.
In respect to results shown in Fig. 4.12 to 4.16, Figures (a) depict the final outcome of conventional SIMP implementation, Figures (b) show the output of the DBN (end of Phase I) while Figures (c) represent the final output of DL-TOP methodology after SIMP fine-tuning step (end of Phase II).

DLTOP performance for twelve classes

In order to evaluate the performance of DL-TOP methodology with reference to its parameters, the basis of the comparison needs to be described. As the scope of DL-TOP is the improvement of computational efficiency of STO, the number of iterations required by the original SIMP for solving each problem represents the basis of comparison, while the iterations required by the DL-TOP methodology are those required to feed the calibrated DBN (part of Phase I) plus those needed by SIMP in Phase II of the methodology. The

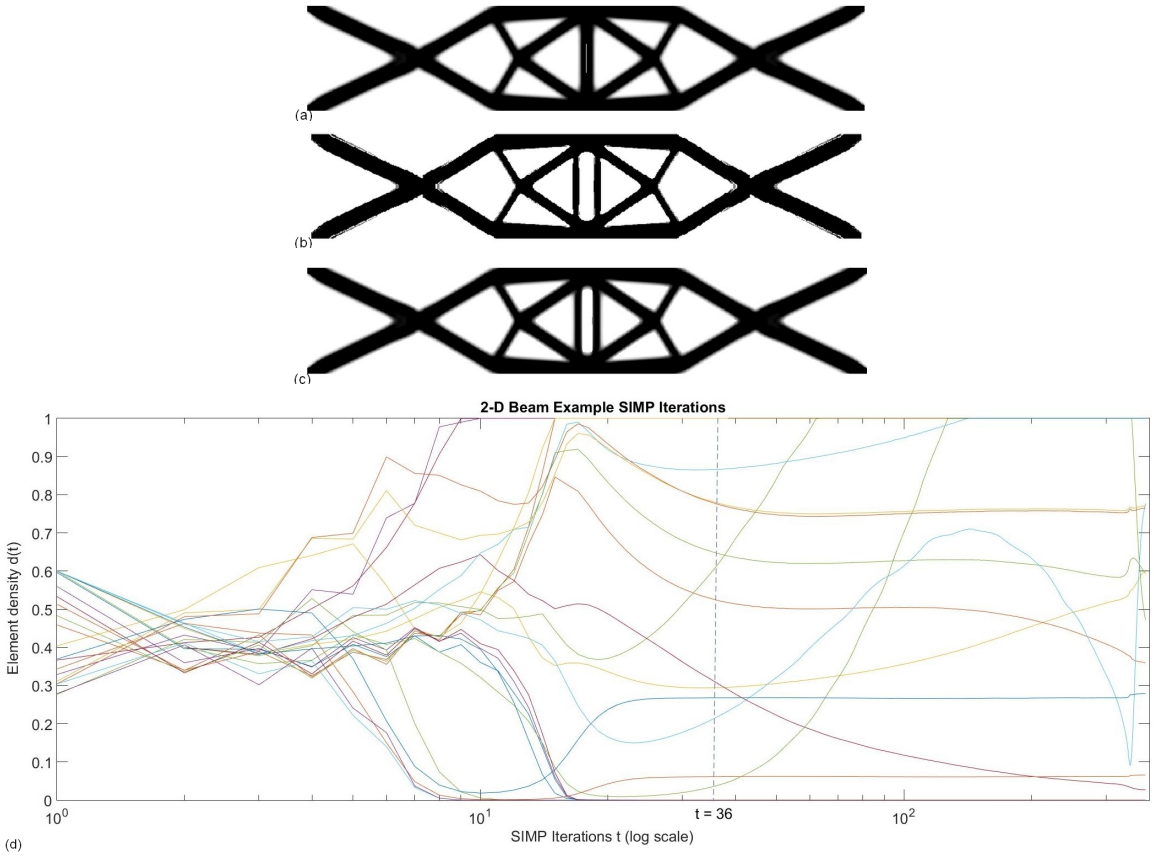


Fig. 4.12 Optimized domain for the short-beam (fine) test example-classification twelve: (a) original SIMP (objective function: 85.99 - iterations: 372), (b) DL-TOP Phase I (objective function: 86.43 - iterations: 36), (c) DL-TOP Phase II (objective function: 85.86 - iterations: 43) and (d) density histories of selected finite elements located in the center of the domain.

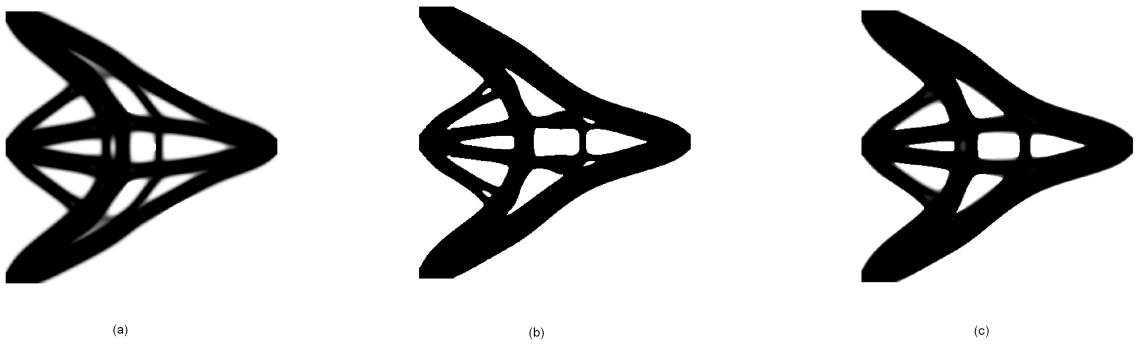


Fig. 4.13 Optimized domain for the antisymmetric test example-classification three: (a) original SIMP (objective function: 22.64 - iterations: 599), (b) DL-TOP Phase I (objective function: 22.95 - iterations: 36) and (c) DL-TOP Phase II (objective function: 22.18 - iterations: 37).

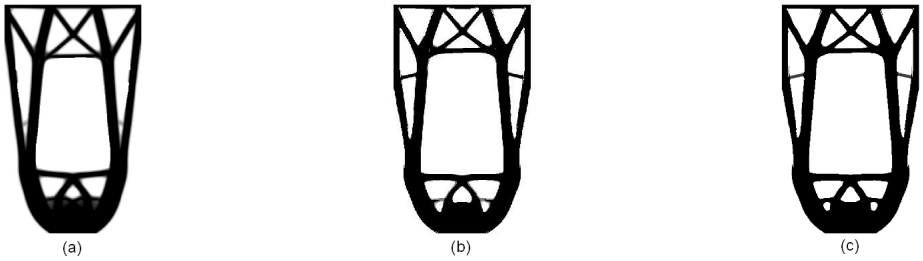


Fig. 4.14 Optimized domain for the column test example-classification three: (a) original SIMP (objective function: 145.12 - iterations: 551), (b) DL-TOP Phase I (objective function: 149.18 - iterations: 36) and (c) DL-TOP Phase II (objective function: 140.90 - iterations: 15).

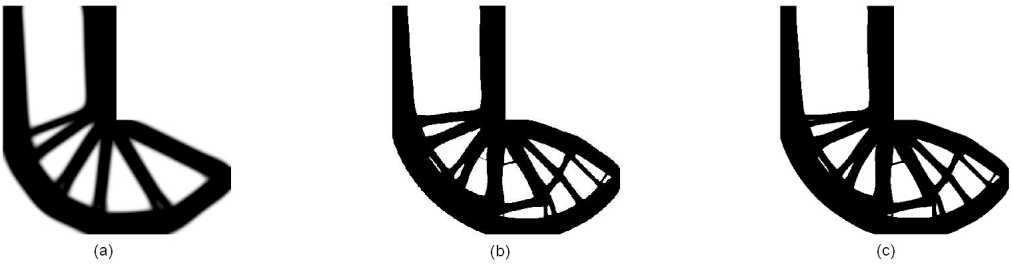


Fig. 4.15 Optimized domain for the L-shape test example-classification three: (a) original SIMP (objective function: 73.06 - iterations: 186), (b) DL-TOP Phase I (objective function: 71.92 - iterations: 36) and (c) DL-TOP Phase II (objective function: 71.19 - iterations: 25).

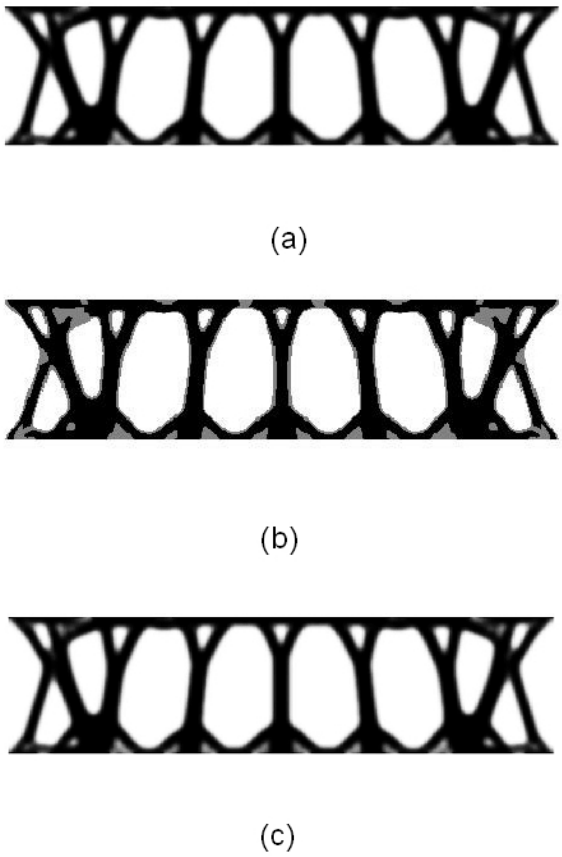


Fig. 4.16 Optimized domain for the long-beam test example-classification three: (a) original SIMP (objective function: 576936.84 - iterations: 775), (b) DL-TOP Phase I (objective function: 479,880.00 - iterations: 36) and (c) DL-TOP Phase II (objective function: 577,826.89 - iterations: 275).

comparison with reference to the computational performance between the original SIMP and DL-TOP is presented in Fig. 4.8. Secondly the original approaches versus the proposed one are also compared with respect to the objective function achieved, this is shown in Fig. 4.9, while Tables 4.2 and 4.3 also show the computational efficiency and robustness of DL-TOP as the average number and variation of iterations required and objective function achieved are presented. Figures 4.8 and 4.9 along with Tables 4.2 and 4.3 present the computational performance and robustness of the proposed DL-TOP methodology for the short-beam (fine) test example. As it can be observe from Table 4.2, SB training dataset on average achieved 83% reduction on the SIMP iterations; the iterations are reduced by almost one order of magnitude from 370 iterations originally required to only 64 iterations on average (the corresponding coefficient of variation (COV) is equal to 16%); while the maximum reduction of the iterations is equal to 87% (see Fig. 4.8(a)). Accordingly, CB training dataset achieved on average 70% reduction of SIMP iterations, while the maximum reduction of SIMP iterations is equal to 77% and COV is equal to 38%. With respect to the objective function value achieved, as it can be seen from Table 4.2, with respect to the training parameters sets on average 1.15% lower value was obtained compared to the one originally achieved by SIMP; correspondingly the objective function value obtained when CB dataset was used is practically equal to the original one.

The optimized domain for the short-beam (fine) test example resulted originally by SIMP is shown in Fig. 4.12(a) and those obtained from Phases I and II of the proposed DL-TOP methodology are depicted in Fig. 4.12(b) and 4.12(c), respectively. As it can be seen the shapes obtained are very similar, while the corresponding objective function values achieved and iterations required are 85.99 and 372, 86.43 and 36, 85.86 and 43 for original SIMP, Phases I and II of DL-TOP, respectively. While the density histories of elements in the centre of the domain are shown in Fig. 4.12(d), where it can be seen the density values of elements' history do not vary monotonously. In Fig. 4.12(d) it can also be noticed that the proposed methodology is able to identify those elements whose density tends to be reduced when approaching the 36_{th} iteration (i.e. purple and orange density lines) of SIMP and generate the hole in the center of the domain of the DL-TOP output. Accordingly, the optimized domains for the antisymmetric, column, L-shape and long-beam test example resulted originally by SIMP and those obtained from Phases I and II of the proposed DL-TOP methodology are depicted in Fig. 4.13to 4.16, respectively. The performance of DL-TOP methodology for the rest of the test examples has a similar performance; more specifically, for the SB training dataset (see Table 4.2 and Fig. 4.8 and 4.9), on average the reduction of SIMP iterations varies from 24% to 73% while the maximum reduction achieved for all these test cases exceeds 80%, the corresponding objective function value achieved on average is slightly

reduced.
In the case of CB training dataset (see Table 4.3 and Fig. 4.8 and 4.9), on average the reduction of SIMP iterations exceeds 15% while the maximum reduction achieved for all these test cases exceeds 60%, the corresponding objective function value achieved on average is also slightly reduced. In general, it should be noted that DL-TOP methodology resulted for all test cases examined to significant decrease of iterations and with the most proper selection of RBM training parameters the decrease exceeds 90%. It is also noticeable that the objective function value is unaffected by the training parameters values and achieving values similar to plain SIMP application. It was also observed that in the case of classification 12, CB training dataset is outperformed by the SB one both in terms of computational efficiency (reduction of SIMP iterations) and robustness.

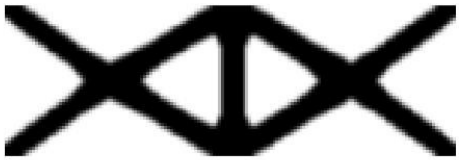
DLTOP performance for three classes

Similar parametric study is performed for the case of three classes (see Fig. 4.10 and 4.11, Tables 4.4 and 4.5); where it is observed that in the short-beam (fine) test example (Fig. 4.10(a) and 4.10(a)), the SB training dataset achieved on average 85% reduction of SIMP iterations and the maximum one is equal to 87% (COV equal to 21%) while the objective function value achieved is on average 2.31% lower to that originally obtained by SIMP approach. The CB training dataset achieved on average 83% reduction of iterations with a maximum reduction equal to 86% (COV equal to 12%) while the objective function value is on average reduced by 3.48%. Accordingly, the performance of DL-TOP methodology for the rest of the test examples has a similar outcome; more specifically, for the SB training dataset (see Table 4.4 and Figures 4.10 and 4.11), on average the reduction of SIMP iterations varies from 55% to 91% while the maximum reduction achieved for all these test cases exceeds 91%, the corresponding objective function value achieved on average is slightly increased.
In the case of CB training dataset (see Table 4.3 and Figures 4.8 and 4.9), on average the reduction of SIMP iterations varies from 72% to 84% while the maximum reduction achieved for all these test cases exceeds 85%; the corresponding objective function value achieved on average is also slightly increased. The optimized domain for the antisymmetric test example resulted originally by SIMP is shown in Fig. 4.13(a) and those obtained from Phases I and II of the proposed DL-TOP methodology are depicted in Fig. 4.13(b) and 4.13(c), respectively. As it can be seen the forms obtained are almost identical, while the corresponding objective function values achieved and iterations required are equal to 22.64 and 599, 22.95 and 36, 22.18 and 37 for original SIMP, Phases I and II of DL-TOP, respectively. Summarizing the results obtained for the case of classification 3, it becomes noticeable that DL-TOP

methodology for all test examples examined resulted to significant reduction of the iterations depicting also remarkable performance stability with reference to the training parameters values. It is also worth noticing that the objective function value is not influenced by these parameters. In classification 3, SB training dataset performed better compared to CB one in terms of iterations decrease but not in the case of objective function value and robustness where CB database performed better than the SB one.
Comparing the classification 3 with the 12 one it can be observed that the latter one is outperformed by the first one both in terms of computational efficiency (reduction of SIMP iterations) and robustness for both training datasets considered. In an attempt to explain this result, it must be pointed out that the classification three procedure is significantly less demanding in terms of DBN training while the classification 12 is not. Additionally, it should be stated that, although it is not guaranteed that the DBN results satisfy the volume constraint, its violation is very limited and a single step of SIMP in Phase II is adequate to correct the required volume fraction often leading to reduction of the objective function value (e.g. compliance) compared to the typical SIMP implementation. As a reference it is noticeable that for a volume fraction equal to 40% the predicted domain achieved for the short-beam (fine) test example is equal to 38.00%, for the antisymmetric one is equal to 39.42%, for the column one is equal to 38.70%, for the L-shape one is equal to 39.74% and for the long-beam one is equal to 42.30%; the corresponding domains are those of Fig. 4.12(b) to 4.16(b), respectively.

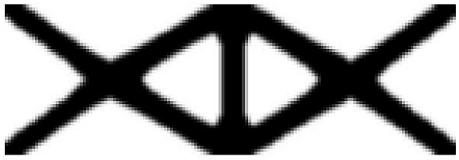
DL-TOP performance in 2D test examples

In order to further evaluate the performance of DL-TOP methodology, three additional 2D test examples are examined using the parameters that were identified in the previous section. The first test example is presented to demonstrate the capabilities of the proposed methodology regarding different update schemes as in this example, the MMA algorithm is used.
The example used is named “short-beam (coarse) test example” described in previous section (see Fig. 4.7(a)) using coarser FE mesh discretization, the sensitivity filter radius changed to two elements and an additional density filter with radius of two elements as well. The new discretization along the x and y axes is equal to $ne_x = 150$ and $ne_y = 50$, whereas the results obtained are shown in Fig. 4.17. The next test example is inspired from the 2-bar problem presented in [194]. The discretization along the x and y axis is equal to $ne_x = 50$ and $ne_y = 20$, respectively, the support conditions refer to fully fixed boundary conditions along the x axis at the base (Fig. 4.18(a)) and the single loading condition refers to one concentrated forces P along the x axis and applied in the middle of the span of the x dimension as depicted in Fig. 4.18(a). The preference for final volume is equal to 20% of the initial domain and



(a)

(b)



(c)

Fig. 4.17 Optimized domain for the short-beam (coarse) test example-MMA: (a) original SIMP (objective function: 30.34 - iterations: 91), (b) DL-TOP Phase I (objective function: 29.77 - iterations: 36) and (c) DL-TOP Phase II (objective function: 30.33 - iterations: 31).

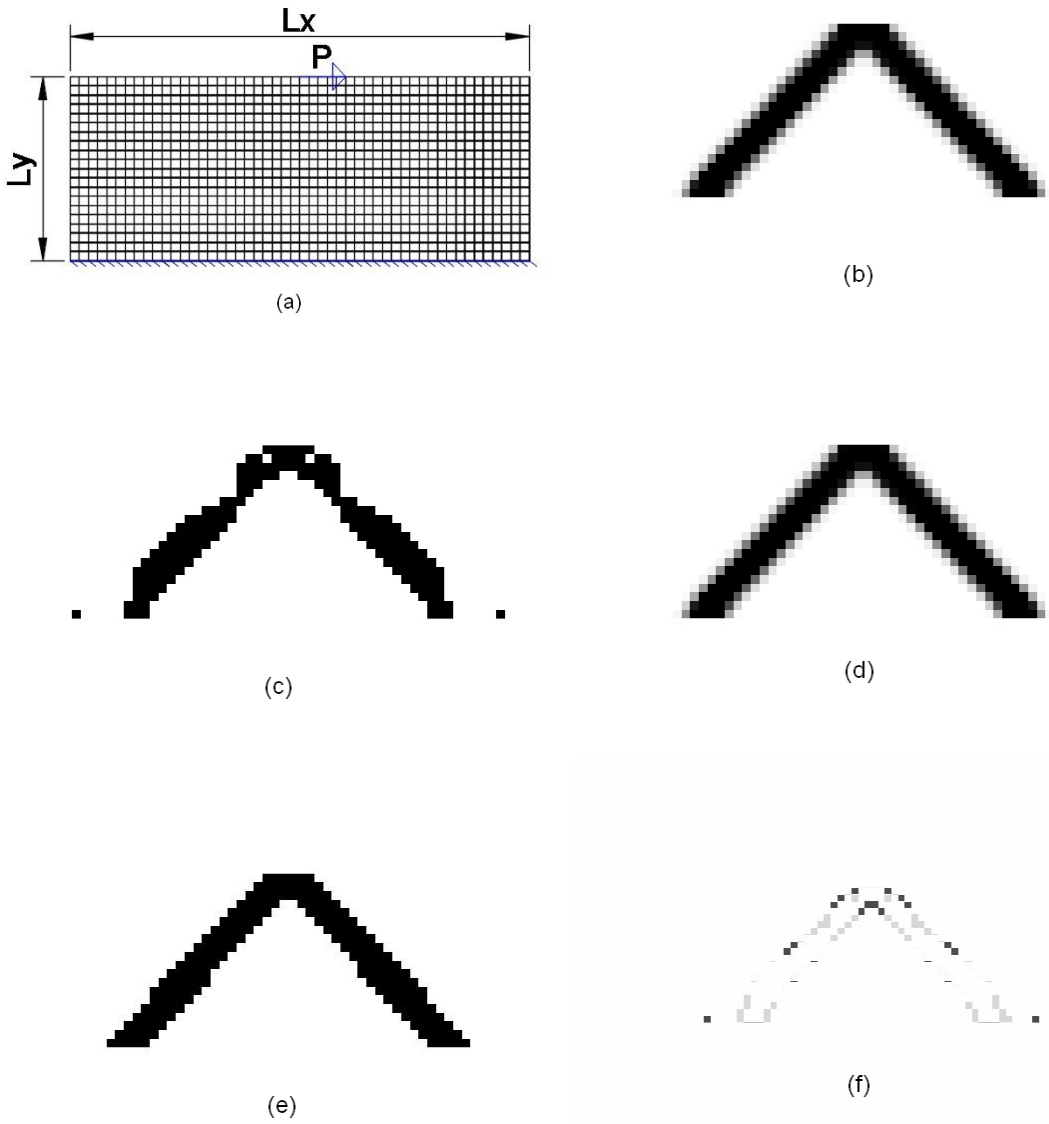


Fig. 4.18 Optimized domain for the 2-bar test example-classification three: (a) original domain, (b) original SIMP (objective function: 10.31 - iterations: 54), (c) DL-TOP Phase I (objective function: 13.33 - iterations: 5), (d) DL-TOP Phase II (objective function: 10.31 - iterations: 28), (e) original SIMP with threshold (objective function: 25.16 - iterations: 5) and (f) difference between DL-TOP and SIMP with threshold.

the filter applied is a density and sensitivity filter with radius equal to 1.5 elements; the results obtained are shown in Fig. 4.18(b) to 4.18(f). The last test example corresponds to the serpentine beam problem presented in the PolyTop [206]. In particular, it corresponds to a non-regular design domain discretized with unstructured polygonal finite element mesh composed by 5,000 elements, the discretization was generated using PolyMesher [205], the support conditions refer to fully fixed boundary conditions along the y axis on the left side of the domain (see Fig. 4.19(a)) and the single loading condition refers to one concentrated force P along the y axis and applied in the pick of the span of the y dimension as depicted in Fig. 4.19(a). The preference for final volume is equal to 40% of the initial domain and the filter applied is a density filter as described in [206] with radius equal to 0.25; the results obtained are shown in Fig. 4.19(b) to 4.19(d). The results obtained when implementing

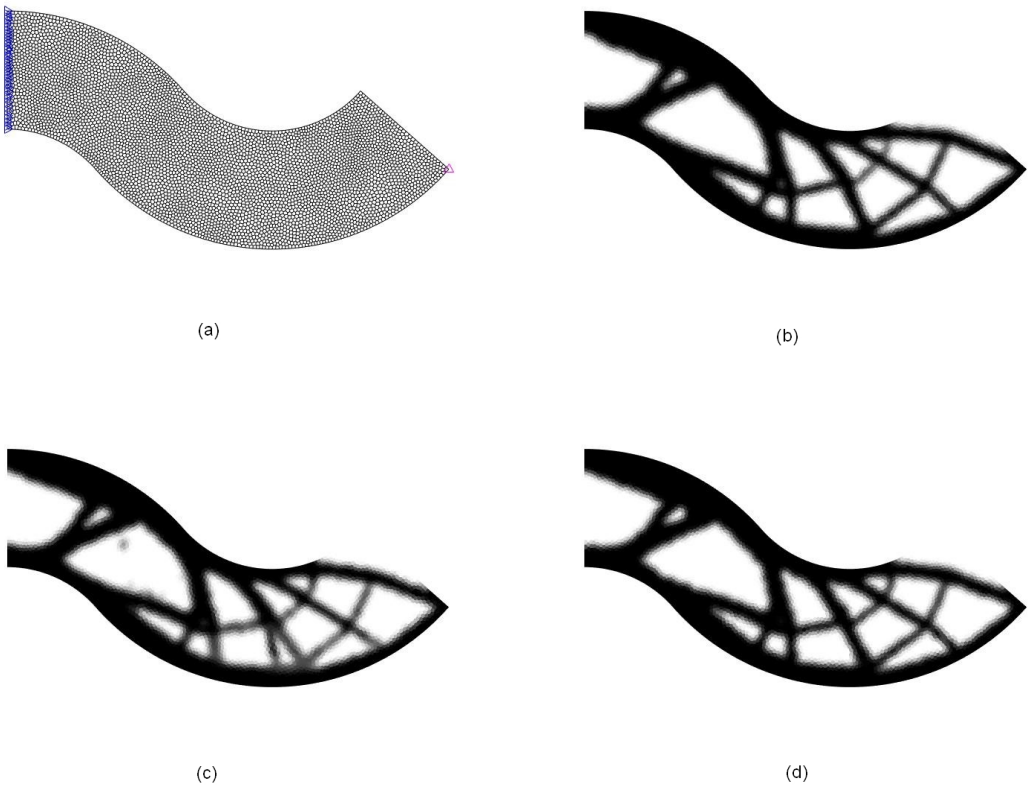


Fig. 4.19 Optimized domain for the serpentine beam test example-classification three: (a) original domain [206], (b) original SIMP (objective function: to 391.19 - iterations: 267), (c) DL-TOP Phase I (objective function: 442.65 - iterations: 36) and (d) DL-TOP Phase II (objective function: 394.46 - iterations: 30).

DL-TOP methodology for the above described three test examples are presented in Table 4.6.

Table 4.6 Average and variance of performance in 2D and 3D test examples.

Test Example	Iterations			Objective function value		
		DLTOP			DLTOP	
	SIMP	Iterations	Variation (%)	SIMP	Value	DLTOP-SIMP Difference (%)
Short-beam (coarse)	91	67	-26.37	30.34	30.33	-0.03
2-bar	54	33	-38.89	10.31	10.31	0
Serpentine beam	267	66	-75.28	391.19	394.46	0.84
L-shape 3D	660	129	-81	15.33	15.65	2.09
Bridge 3D	509	193	-62.08	1,632,305.73	1,640,121.40	0.48

In particular, regarding the short-beam (coarse) test example, DL-TOP achieved reduction of more than 25% on the SIMP iterations required originally and the objective function value achieved is more or less equal to that obtained by SIMP approach. Accordingly, for the 2-bar test example DL-TOP achieved almost 40% reduction on SIMP iterations and the objective function value achieved is equal to that originally obtained by the SIMP.
Additionally, in the 2-bar test example, a threshold was applied on the result of SIMP achieved after performing the same number of iterations with those used as input by DL-TOP in order to witness the differences in these two applications. The result obtained by using a thresh-old is presented in Fig. 4.18(e), the result obtained by DL-TOP Phase I is presented in Fig. 4.18(c) while the difference between these two results is shown in Fig. 4.18(f). In Fig. 4.18(f), black areas denote the elements present in the DL-TOP Phase I and not in the threshold and grey areas are the elements present in the threshold and not in the DL-TOP Phase I. According to DL-TOP methodology material has been added in the outside areas of both “legs” while it has also removed plenty of material from the inner areas as well. This can be explained as DL-TOP has identified the tendency of these density values to increase and decrease accordingly, leading to a result closer to the final one of just implementing SIMP (i.e. Fig. 4.18(b), 54 iterations). It can be witnessed that in the serpentine beam test example DL-TOP achieved more than 75% reduction on the SIMP iterations required originally and the objective function value achieved is more or less equal to that obtained by SIMP approach. The optimized domains for the all three test example are shown in Fig. 4.17 to 4.19 as well as the objective functions and compliance for the original topology optimization and Phases I and II of the DL-TOP.

DL-TOP performance in 3D test examples

Given the performance evaluation of the training parameters combination for the DBN part of the DL-TOP methodology, its computational efficiency is also assessed over two 3D topology

optimization test examples. The first one shown in Fig. 4.20(a) is three-dimensional version of the L-shape test example, where the discretization along the x, y and z axes is taken equal to 60, 60 and 20, resulting into 72,000 solid FEs, respectively, the support conditions refer to fully fixed boundary conditions for the xz plane along the z dimension, ending at the $\frac{1}{3}rd$ of the x dimension (Fig. 4.20(a)) and the loading condition refers to a concentrated force P along the y axis applied at the $\frac{1}{4}th$ of the y dimension and the middle of the span along the z dimension. The second 3D test example also shown Fig. 4.20(b) was taken from the example

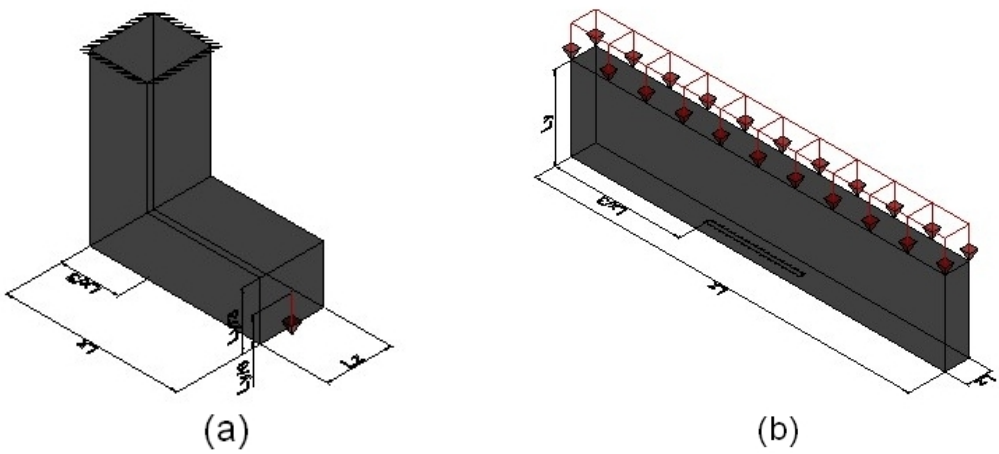


Fig. 4.20 3D test examples: (a) L-shape 3D and (b) bridge.

examined in [101]. The discretization along the x, y and z axes is taken equal to 160, 40 and 13, respectively, resulting into 83,200 solid finite elements, the support conditions refer to fully fixed support at the xz plane spanning from the $\frac{1}{3}$ to the $\frac{1.75}{3}$ of the x dimension and one element in each size from the center of the z dimension and the loading conditions refer to distributed loading q along that is applied on the top xz plane along the y dimension (as show in Fig. 4.20(b)).

In the first 3D test example the preference for final volume is equal to 15% of the initial cubic domain and the filter radius is equal to 1.2 elements. In the second 3D test example the preference for the final volume is equal to 40% of the initial domain and the filter radius is equal to 1.5 elements. For both test examples filtering was implemented using a combination of the standard sensitivity filtering with the density one. The results obtained when implementing DL-TOP methodology are presented in Table 4.6. In particular, regarding the L-shape 3D test example DL-TOP achieved 81% reduction on the SIMP iterations required originally and the objective function value achieved is basically equal to that obtained by SIMP approach. Accordingly, for the bridge 3D test example DL-TOP achieved 62% reduction of SIMP iterations and the objective function value achieved is equal to that originally obtained by

the SIMP. The optimized domain for the bridge 3D test example resulted originally by SIMP is shown in Fig. 4.21(a) and those obtained from Phases I and II of the proposed DL-TOP methodology are depicted in Fig. 4.21(b) and 4.21(c), respectively. As it can be seen the forms obtained are almost identical, while the corresponding objective function values achieved and iterations required are equal to 1,632,305.73 and 509, 1,612,500.00 and 36, 1,640,121.4 and 193 for original SIMP, Phases I and II of DL-TOP, respectively. In addition, the optimized domain of the L-shape 3D example can be seen in Fig. 4.22 as well as the objective functions and compliance for the original topology optimization and Phases I and II of DL-TOP.

The computer hardware platform that was used for the purposes of these tests consists of an Intel Xeon E5-1620 at 3.70 GHz quad-core (with 8 threads) with 16 GB RAM for the case of CPU based computations and NVIDIA GeForce 640 with 384 cores and 2 GB RAM for the case of GPGPU based computations and the operating system was Windows 10 (64bit). For the bridge 3D test example both sequential and parallel test runs are carried out, and as it can be seen the computing time required for solving the topology optimization problem discretized with 83,200 solid finite elements SIMP requires up to 54,287 seconds to carry out 509 SIMP iterations while applying a GPGPU-based acceleration of the structural analysis procedure the re-quired time is reduced to 8,587 seconds (speedup factor of 6× for GPU when compared with CPU). The time required by the proposed DL-TOP methodology is 20,910 seconds if the structural analysis part of the SIMP iterations is performed sequentially, while if a GPGPU-based acceleration of the structural analysis is performed the computing time is further reduced to 3,256 seconds (speedup factor of 17× for DL-TOP-GPU when compared with SIMP-CPU).

Worth mentioning that the computational time required for training the network is not included in the time requirements described above as the network was not trained on the specific examples or on any of the test examples presented in the current study. Training of the network is performed only once based on a database derived using a single test example, the resulting metamodel through training is unique and is used for any 2D or 3D test examples without requiring additional training. For this reason, it would be misleading to add the training time required ones to the comparison between SIMP and DL-TOP for every test example. Training is performed once and the resulting metamodel can be applied to any topology optimization problems.

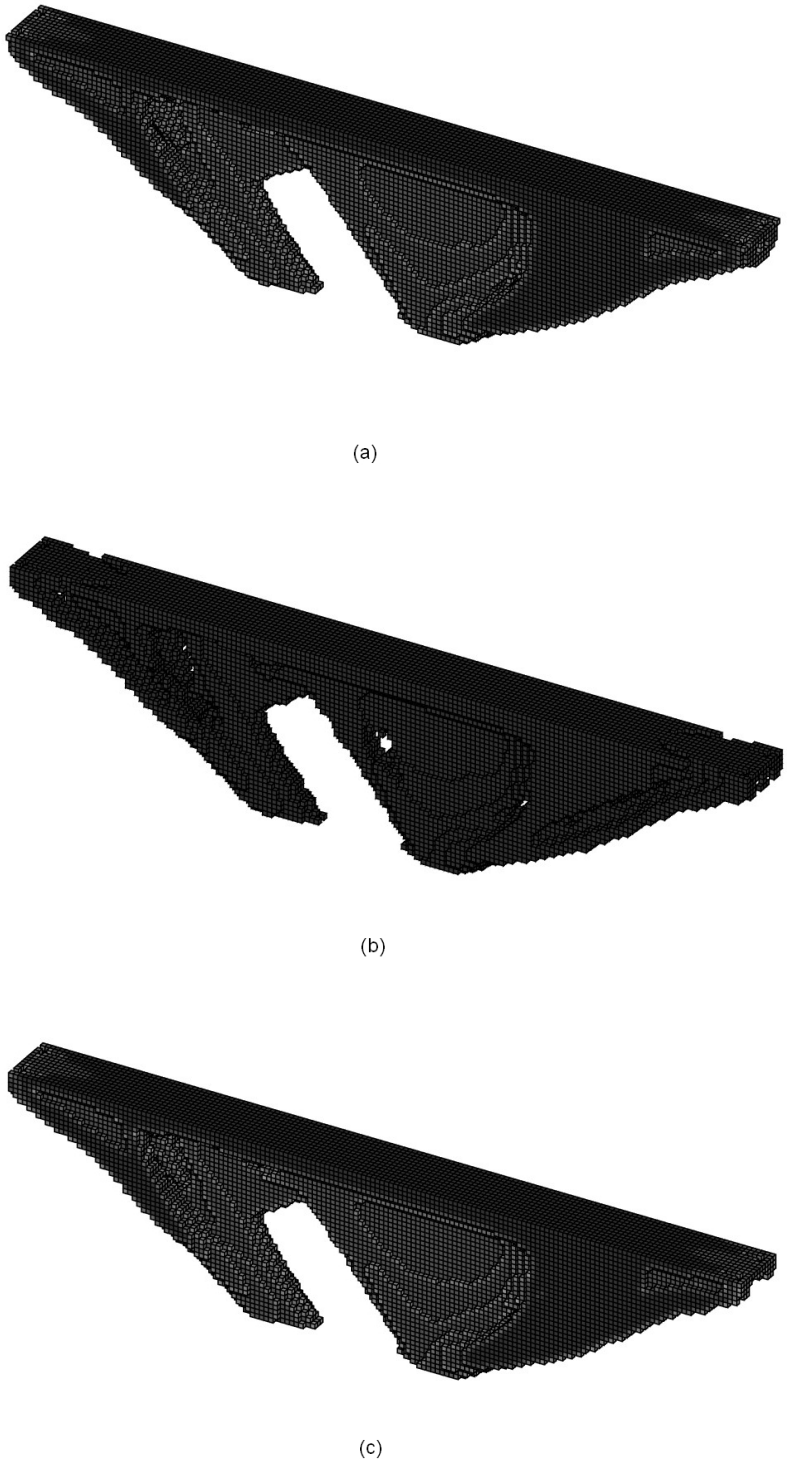


Fig. 4.21 Optimized domain for the 3D bridge test example-classification three: (a) original SIMP (objective function: to 1,632,305.73 - iterations: 509), (b) DL-TOP Phase I (objective function: 1,612,500.00 - iterations: 36) and (c) DL-TOP Phase II (objective function: 1,640,121.40 - iterations: 193).

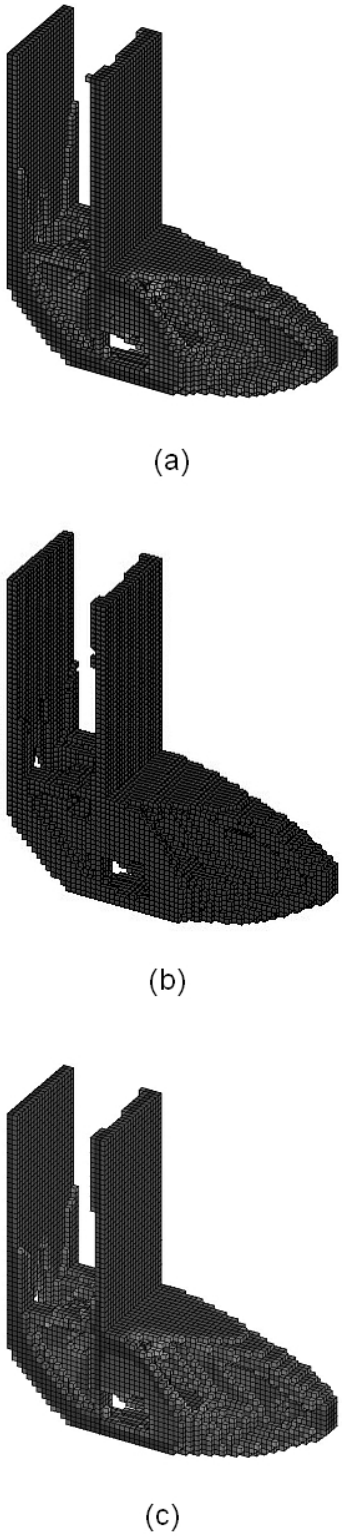


Fig. 4.22 Optimized domain for the 3D L-shape test example-classification three: (a) original SIMP (objective function: 15.33 - iterations: 660), (b) DL-TOP Phase I (objective function: 13.36 - iterations: 36) and (c) DL-TOP Phase II (objective function: 15.65 - iterations: 93).

Chapter 5

Deep learning in reduced order modeling

This chapter focuses on exploring new methods of reduced order modeling for minimizing the computational needs of topology optimization (TO) approaches. All TO approaches are computationally demanding especially when detail level raises. Although the available computing power is constantly improving, either in Central Processing Unit (CPU) or Graphical Processing Unit (GPU) in terms of speed and memory per core and number of cores, the sizes and complexity of TO problems remain time and computational load greedy. As this drawback concerns practically everyone who works in the field of TO, several approaches have been proposed for reducing the necessary execution time of TO mainly via incorporating parallel programming in CPU or GPU [16, 24, 45, 138]. In this part of the thesis, the possibility of developing new methods for TO, focused on accelerating the TO procedure by dramatically reducing its computational cost through reduced order modeling is examined. The implemented research focused on taking advantage of the capabilities of up-to-date deep learning methods in combination with established TO methodologies. In detail, three new methods are presented:

1. DL-SCALE
2. DLRM-TOP
3. CN-TOP

These three methods can be regarded as new reduced order modeling methods, each one with a different architecture described further along. All the above methods present significant reduction of computational loads in comparison with typical TO methods.

5.1 DL-SCALE - Deep Learning Assisted Model Upgrading

The advanced performance observed by DL-TOP in accelerating topology optimization application, led us to think additional possibilities to exploit its capabilities on a different framework. In the previous section, the ability of DBN networks to discover non-linear correlations between initial values of densities and the final value per FE as calculated by SIMP was proven. The exploitation of data was based on the principle that density fluctuation of early SIMP iterations can act as classification input of a DBN with respect to final density value.
By incorporating the above features in a model upgrading scheme, a new methodology for performing topology optimization (DL-SCALE) is proposed. This methodology is based on deep belief networks and SIMP, formulated in a sequential "model optimize and upgrade" architecture which is analytically described in the following section.

5.1.1 DL-SCALE methodology description

Computing time requirements of topology optimization problems regardless of available computing power depend heavily on the number of finite elements that are used for the mesh discretization. This dependency is bound to the "curse of dimensionality" as topology optimization applications are based on iterative solutions per finite element. Several computational approaches have been proposed for reducing the execution time per iteration through parallel programming and exploitation of the processors present in CPU and/or GPU [16, 24, 45, 138]. The proposed methodology is focused on using low-cost results of DL assisted TO results of "sparsely" discretized domains for reducing the computational time of much denser meshes.
DL-SCALE is formulated as a multi-stage repetitive combination of a number of DL-TOP phases. As presented in the previous section of this dissertation, DL-TOP can dramatically reduce the number of iterations needed by SIMP for solving a structural topology optimization problem. A small number of iterations of SIMP are fed into a trained DBN network which predict a close-to-final optimized model while this model is fine-tuned by SIMP. The idea that generated DL-SCALE, is based on combining low computational times of sparsely meshed domains in SIMP approach and iteration reduction capabilities of DL-TOP methodology. By combining the above, a significant reduction of computational times of densely meshed domains is achieved as it shown in test examples presented in the following sections.
A topology optimization problem TOP_0 can be characterized by the following parameters:

- Domain dimensions L_{xyz}
- Population of FEs in the mesh ne
- Support conditions S_c
- Loading conditions P_c
- Desired volume V_t

Accordingly, the above parameters of TOP_0 can be described as $[L_{xyz}^{(0)}, ne^{(0)}, S_c^{(0)}, P_c^{(0)}, V_t^{(0)}]$. The execution time that SIMP needs for solving TOP_0 is equal to t_0. It can be stated that the execution time for solving TOP_1 with parameters $[L_{xyz}^{(0)}, ne^{(1)}, S_c^{(0)}, P_c^{(0)}, V_t^{(0)}]$ with respect to t_0 depends on the ratio: $\frac{ne^{(0)}}{ne^{(1)}}$. A small example of this can be seen in Table 5.1 where the increase of computational time per iteration for different mesh densities is described. The TO example used is the 3D Bridge example presented in the previous section. The specifications of the computer used are: an Intel Xeon E5-1620 at 3.70 GHz quad-core and 16 GB RAM while the SIMP code used is the 3D version of the 88-line code presented by Andreassen et al. [4]. As witnessed in Table 5.1, mesh density plays an important role in

Table 5.1 Mesh density and execution time ratio for SIMP.

	$ne^{(0)}$			$ne^{(1)}$		
ne	406456	63480	32368	16200	8064	4000
$\frac{ne^{(0)}}{ne^{(1)}}$	1.00	6.40	12.56	25.09	50.40	101.61
$\frac{t^{(0)}}{t^{(1)}}$	1.00	12.89	27.72	59.35	137.00	374.70

computational time of TO problems. More precisely, if we consider t_d as the summation of $t_i^{(1)}\ \forall\, i \in [1,5]$ of the five $ne_i^{(1)}$ mesh discretizations examined, it can be calculated that: $\frac{t^{(0)}}{t^{(d)}} = 7.12$. Thus, performing a number of iterations of sparse meshes is more economical in terms of execution time than performing one iteration of a dense mesh. It is also worth mentioning that in general, sparsely meshed domains require less iterations than densely meshed ones for converging.

DL-SCALE can be described as follows. In case TO is to be performed in a domain meshed in ne_f elements and the parameters of this TOP_f are $[L_{xyz}^{(f)}, ne^{(f)}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}]$, a population of five reduced order models in terms of ne are created as TOP_i^R where $i \in [1,5]$. All the created TOP_i are identical to TOP_f with respect to all parameters except for $ne^{(i)}$. As a result, each of the i_{th} reduced models TOP^R can be described as $[L_{xyz}^{(f)}, ne_i^{(R)}, S_c^{(f)}, P_c^{(f)}, V_t^{(f)}]$.

Regarding discretization densities, it must be pointed out that:

$$ne_1^{(R)} < ne_2^{(R)} < ne_3^{(R)} < ne_4^{(R)} < ne_5^{(R)} << ne_f^{(R)} \tag{5.1}$$

It is worth pointing out that while all parameters for the five $TOP_i^{(R)}$ are defined, the domain D of only the first one is formulated. Since the definition of the five $TOP^{(R)}$ is completed, DL-SCALE application continues with the following procedure. DL-TOP is applied on the $TOP_1^{(R)}$. Hence, SIMP performs 36 iterations on TOP_1^R. The density values per iteration per FE are used by a trained DBN for predicting a final density value for each FE in $TOP_1^{(R)}$ domain. It must be pointed out that since the input and output used in DL-SCALE are identical with the ones used in DL-TOP, the procedure followed for creating a database, training and calibrating the DBN model is exactly the same as the one described in the DL-TOP presentation section. Additionally, the DBN architecture is also identical with the one used in DL-TOP.

Since an optimized domain for $TOP_1^{(R)}$ is defined, a convolution of this domain is executed. This convolution is performed according to Eq. 3.34 while the filter used is a Gaussian blur filter [200]. The dimensions $[a_f, b_f]$ of filter f are chosen with respect to dimension of D ($[ne_x, ne_y]$). An example of values of f for a $[7,7]$ filter can be seen in Eq. 5.2:

$$f = \begin{bmatrix} 0 & 0 & 0 & 0.01961 & 0 & 0 & 0 \\ 0 & 0.01961 & 0.07059 & 0.12549 & 0.07059 & 0.01961 & 0 \\ 0 & 0.07059 & 0.25098 & 0.39216 & 0.25098 & 0.07059 & 0 \\ 0.01961 & 0.12549 & 0.39216 & 0.39216 & 0.39216 & 0.12549 & 0.01961 \\ 0 & 0.07059 & 0.25098 & 0.39216 & 0.25098 & 0.07059 & 0 \\ 0 & 0.01961 & 0.07059 & 0.12549 & 0.07059 & 0.01961 & 0 \\ 0 & 0 & 0 & 0.01961 & 0 & 0 & 0 \end{bmatrix} \tag{5.2}$$

For the convolution step, the size parameters are chosen accordingly in order for the convolved matrix D_c to be equal to the ones of the initial matrix D. Once convolution is performed, the convolved matrix D_c is normalized in the $[0,1]$ range. It is worth pointing out that the above are referring to the case of a 2D domain but the same procedure can be applied to 3D domains as well.

The D_c matrix represents an optimized domain for $TOP_1^{(R)}$. In the next step of DL-SCALE, D_c is re-meshed with the new population of FEs being equal to the ones of $TOP_2^{(R)}$:$ne_2^{(R)}$. As coordinates $x_i^{(1:k)}$, $y_i^{(1:k)}$ of the k edges of each i_{th} FE in $TOP_1^{(R)}$ and $TOP_2^{(R)}$ are known,

the geometric center of each FE can be calculated according to:

$$x_i^c = \frac{1}{k}\sum_{j=1}^{k} x_i^j$$
$$y_i^c = \frac{1}{k}\sum_{j=1}^{k} y_i^j \tag{5.3}$$

The density value of each of the $ne_2^{(R)}$ elements of the $TOP_2^{(R)}$ is taken equal to the density of the closest, in terms of geometric center distance, FE of the $TOP_1^{(R)}$ domain. This projection of the D_c of $TOP_1^{(R)}$ in $TOP_2^{(R)}$ dimensions is regarded as a proposed TO initialization point instead of the uniform density initialization that is commonly used. In the next step of DL-SCALE, the generated domain of $TOP_2^{(R)}$ is passed into SIMP and again a population of $t = 36$ iterations of SIMP are performed for generating an input for a second DBN application. The DBN proposes a final density for each FE in the $TOP_2^{(R)}$ domain. As with $TOP_1^{(R)}$ DBN-proposed model, the $TOP_2^{(R)}$ proposed model is re-meshed to a population of FEs equal to $ne_3^{(R)}$ and the volume of each FE of $TOP_2^{(R)}$ is assigned a value as described in the previous step of DL-SCALE. The same procedure is repeated for $TOP_3^{(R)}$, $TOP_4^{(R)}$ and $TOP_5^{(R)}$. At the final step of DL-SCALE, the DBN-proposed TOP_5 is re-meshed with the new population of FEs being equal to ne_f. Initial densities of each of the ne_f FEs are set equal to the density of the closer $TOP_5^{(R)}$ element with respect to geometrical center distance. After applying convolution, as described previously, the produced output is used as SIMP initialization for TOP_f and SIMP performs 36 needed iterations followed by a DBN-based prediction, as in DL-TOP, of a close-to-final optimized topology. The final output is produced after fine-tuning by SIMP performing the necessary iterations until convergence. A flowchart of DL-SCALE can be seen in Fig. 5.1.

5.1.2 Test examples

In order to evaluate the performance of DL-SCALE methodology, a series of 2D test-examples, known to literature, are used. In detail, five test-cases are solved using SIMP methodology and record is kept regarding necessary iterations for convergence, objective function value (compliance) and execution time. The same test-examples are also optimized with the use of DL-SCALE methodology and the same records are kept. As according to DL-SCALE, iterations are performed on domains with less dense meshes, the comparison is based on execution time. In the case of SIMP, the total time from the domain definition and up to the end of the final iteration is recorder. In the case of DL-SCALE, the time from the definition of $TOP_1^{(R)}$ to the end of the final iteration of TOP_f is recorded.

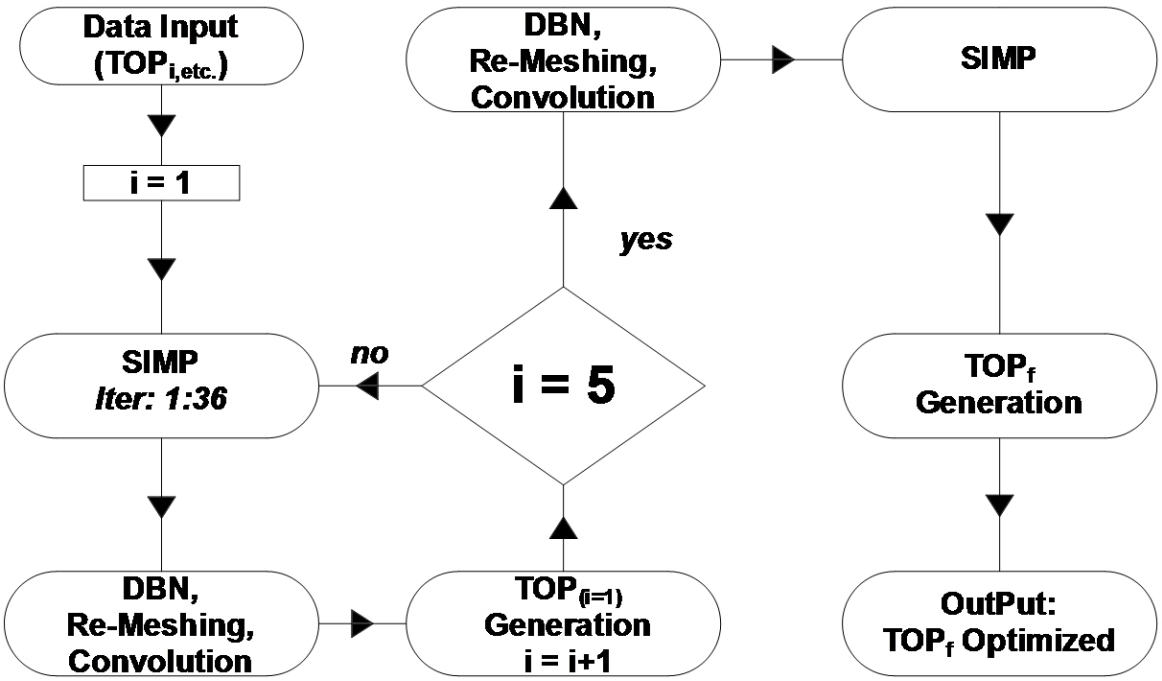


Fig. 5.1 Flowchart of DL-SCALE methodology

The iterations recorded in DL-SCALE are the 36 iterations performed before DBN application along with the ones performed following DBN output until convergence. As DL-SCALE requires the formulation of five domains with reduced population of FEs per test case, two different samples of $[TOP_i^R]$ are used. In the first one, each of the $ne^{(i)}$'s is equal to [1000, 2000, 3000, 4000, 5000] respectively while in the second case, each of the $ne^{(i)}$'s is equal to [3000, 4000, 5000, 7000, 10000] respectively. Regarding the population of FEs in the final domain of each test-example, four different populations are examined in order to evaluate the performance of DL-SCALE. These four populations are: $[20,000,\ 50,000,\ 75,000,\ 100,000]$. The five test-examples used are described bellow.

Test-Examples description

In Test-Example A, the support conditions refer to two fixed joints placed at the two right corners of the domain and the loading conditions refer to two concentrated forces P along the x axis and applied in the left and right middle of the span in the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 40%.

In Test-Example B, the support conditions refer to fully fixed boundary condition along the x axis ending at the half of the x dimension and the loading condition refers to one concentrated force P along the y axis and applied in the fourth of the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 35%.

In Test-Example C, the support conditions refer to fully fixed boundary conditions along the x axis, starting at the 3:8ths of the x dimension and the loading conditions refer to four concentrated forces P along the y axis and applied at intermediate distances equal to 1:3 of the x dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 50%.

In Test-Example D, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y axis, applied in the middle of the y dimension at the base of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 35%.

In Test-Example E, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y, applied at the half of the y dimension at the top of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 35%.

It is worth pointing out that a sensitivity filter with radius equal to 3 was chosen in all cases as SIMP filter. A schematic representation of the test-examples can be seen in Fig. 5.2.

5.1.3 Results

In Test-Example A, DL-SCALE achieved a maximum reduction of computational time equal to 81.73% in the case of 100,000 elements in TOP_f with respect to SIMP while the objective function value was decreased by 0.40%. All data recorded in this test can be seen in Table 5.2. Additionally, the final topologies produced by SIMP alone and the ones proposed by DL-SCALE, per discretization can be seen in Fig 5.3(a) and Fig. 5.3(b) respectively. The shapes produced by the DBN for each discretization and the ones from the convolution stage per discretization can be seen in Fig. 5.4(a) and Fig. 5.4(b) respectively.

Table 5.2 DL-SCALE performance in Test-Example A.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
20,000	299	20.40	105.74	66	20.08	47.50	55.08	1.54
50,000	308	21.24	289.37	73	21.07	95.82	66.89	0.82
75,000	396	21.71	583.18	77	21.53	143.91	75.32	0.85
100,000	439	21.94	882.32	63	21.85	161.19	81.73	0.40

In Test-Example B, DL-SCALE achieved a maximum reduction of computational time equal to 81.96% in the case of 100,000 elements in TOP_f with respect to SIMP while the objective function value was decreased by 0.94%. All data recorded in this test can be seen in Table 5.3. Additionally, the final topologies produced by SIMP alone and the ones proposed by DL-SCALE, per discretization can be seen in Fig 5.5(a) and Fig. 5.5(b) respectively. The

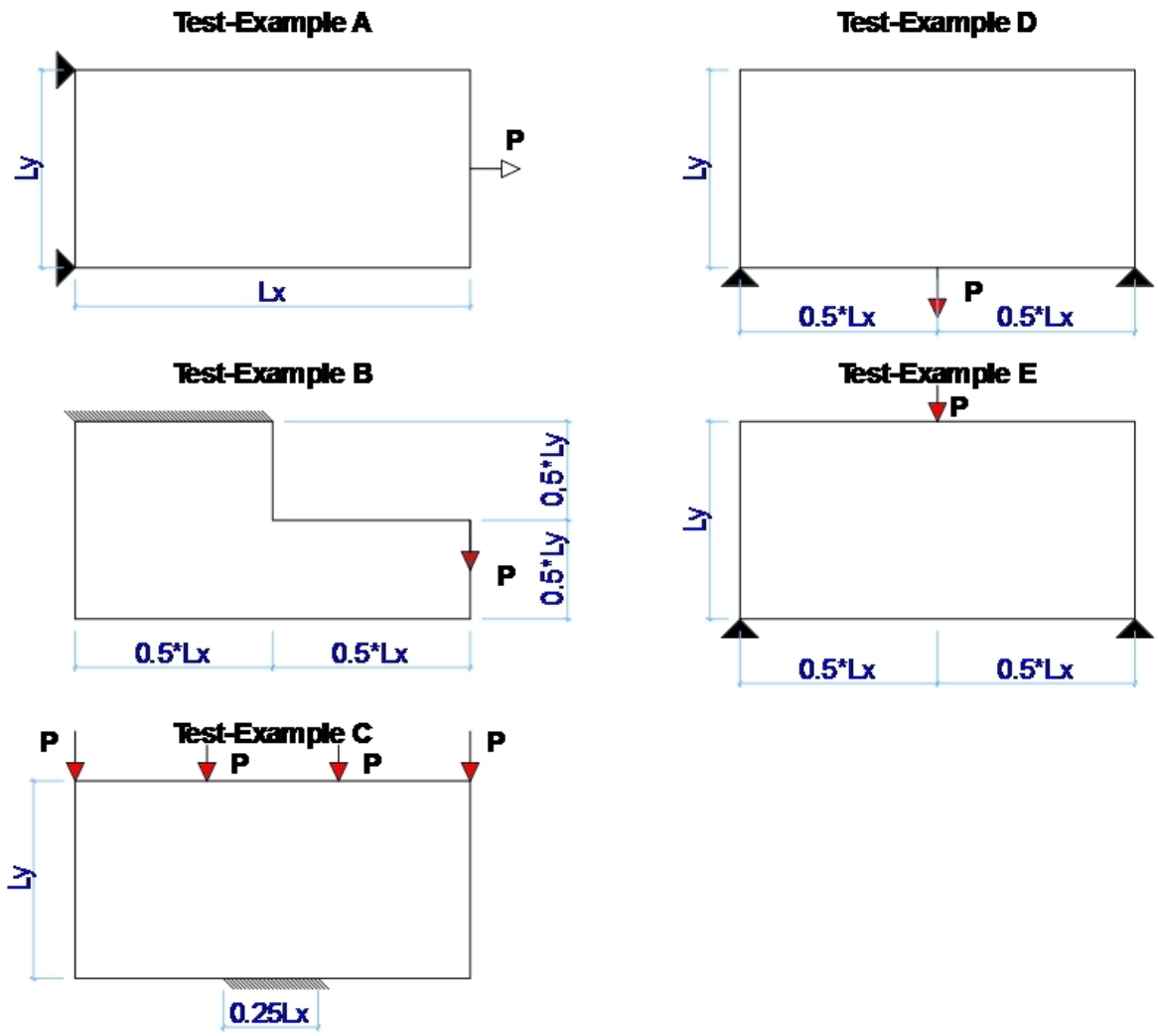


Fig. 5.2 Schematic representation of Test-Examples

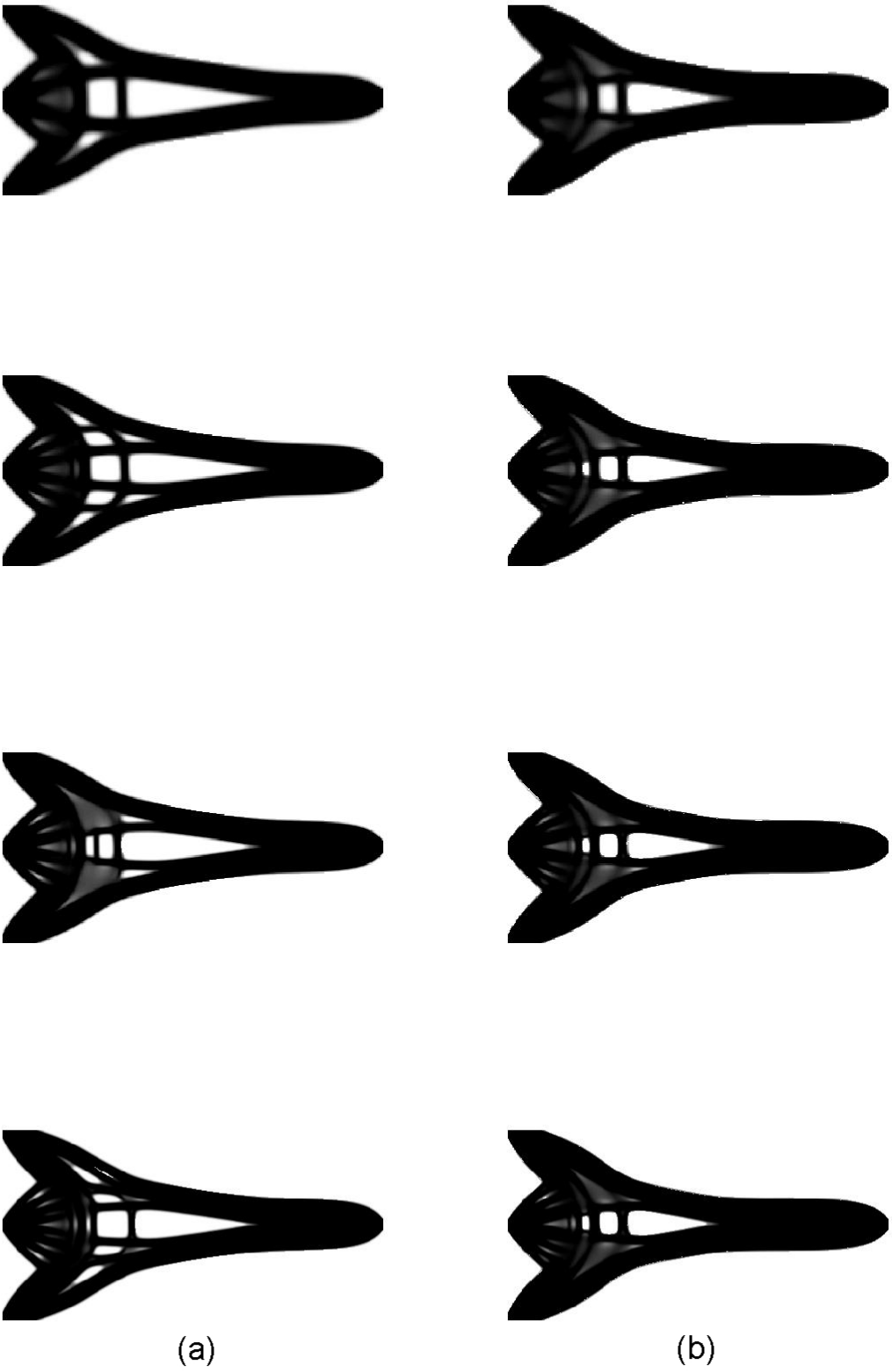


Fig. 5.3 Optimized domain for Test-Example A for each discretization: (a) SIMP output, (b) DL-SCALE output.

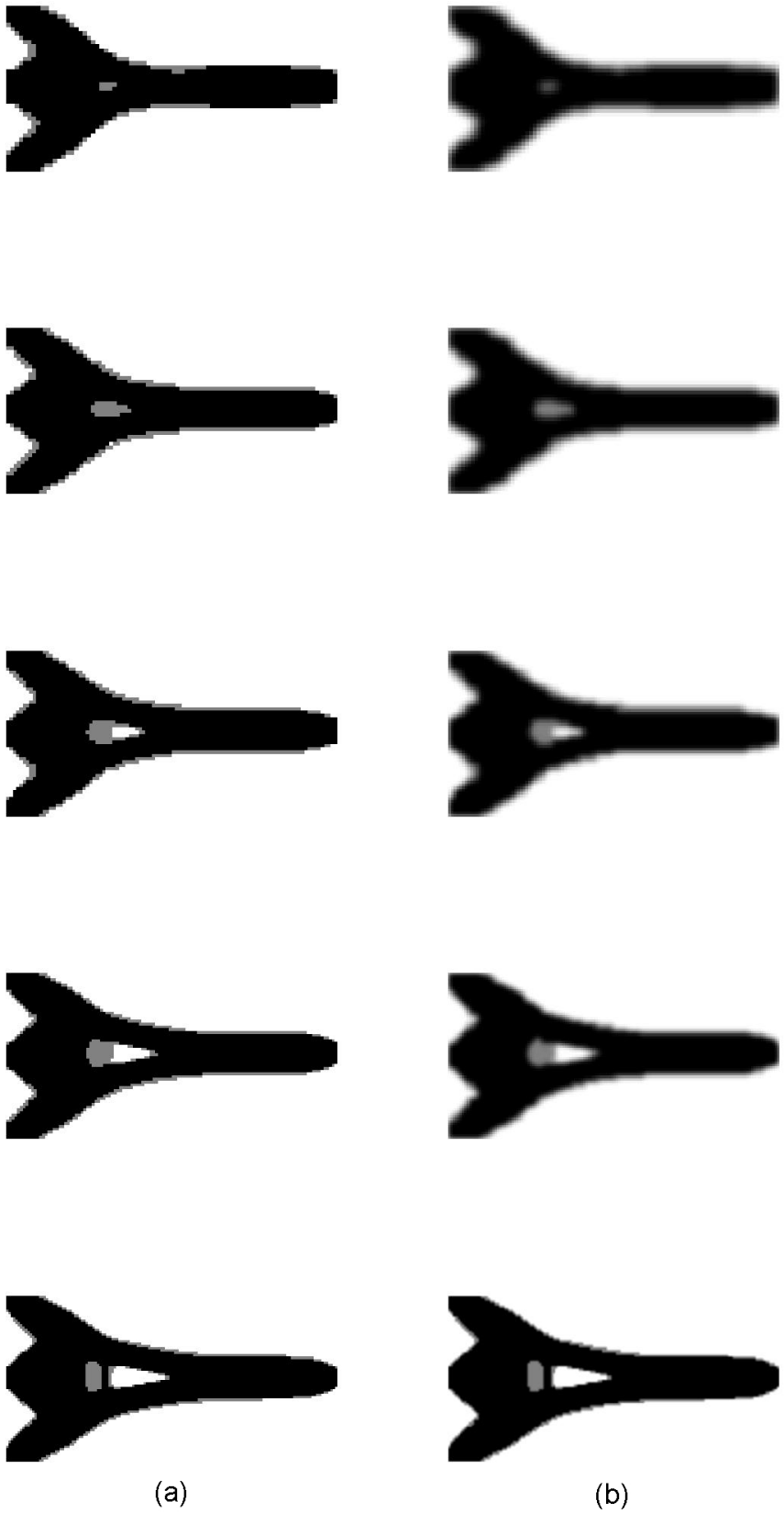


Fig. 5.4 Optimized domains for each discretization of Test-Example A as exported from: (a) DBN, (b) Convolution.

shapes produced by the DBN for each discretization and the ones from the convolution stage per discretization can be seen in Fig. 5.6(a) and Fig. 5.6(b) respectively.

Table 5.3 DL-SCALE performance in Test-Example B.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
20,000	220	144.36	68.03	65	141.31	33.23	51.16	2.11
50,000	322	139.14	259.31	60	136.73	62.32	75.97	1.73
75,000	403	139.46	499.28	59	137.41	88.45	82.28	1.47
100,000	375	138.16	631.16	57	136.86	113.85	81.96	0.94

In Test-Example C, DL-SCALE achieved a maximum reduction of computational time equal to 82.66% in the case of 100,000 elements in TOP_f with respect to SIMP while the objective function value was decreased by 1.65%. All data recorded in this test can be seen in Table 5.4. Additionally, the final topologies produced by SIMP alone and the ones proposed by DL-SCALE, per discretization can be seen in Fig 5.7(a) and Fig. 5.7(b) respectively. The shapes produced by the DBN for each discretization and the ones from the convolution stage per discretization can be seen in Fig. 5.8(a) and Fig. 5.8(b) respectively.

Table 5.4 DL-SCALE performance in Test-Example C.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
20,000	49	110.04	17.27	56	106.40	43.68	-153.01	3.31
50,000	147	109.02	131.83	61	107.03	82.11	37.71	1.83
75,000	131	109.84	183.52	90	108.11	155.90	15.05	1.58
100,000	438	111.53	845.44	59	109.70	146.61	82.66	1.65

In Test-Example D, DL-SCALE achieved a maximum reduction of computational time equal to 76.60% in the case of 100,000 elements in TOP_f with respect to SIMP while the objective function value was decreased by 0.87%. All data recorded in this test can be seen in Table 5.5. Additionally, the final topologies produced by SIMP alone and the ones proposed by DL-SCALE, per discretization can be seen in Fig 5.9(a) and Fig. 5.9(b) respectively. The shapes produced by the DBN for each discretization and the ones from the convolution stage per discretization can be seen in Fig. 5.10(a) and Fig. 5.10(b) respectively.

Table 5.5 DL-SCALE performance in Test-Example D.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
20,000	219	18.48	69.70	53	18.13	30.13	56.77	1.89
50,000	245	19.00	198.54	64	18.77	68.42	65.54	1.21
75,000	277	19.22	342.51	64	18.99	100.65	70.62	1.20
100,000	321	19.38	540.44	60	19.21	126.45	76.60	0.87

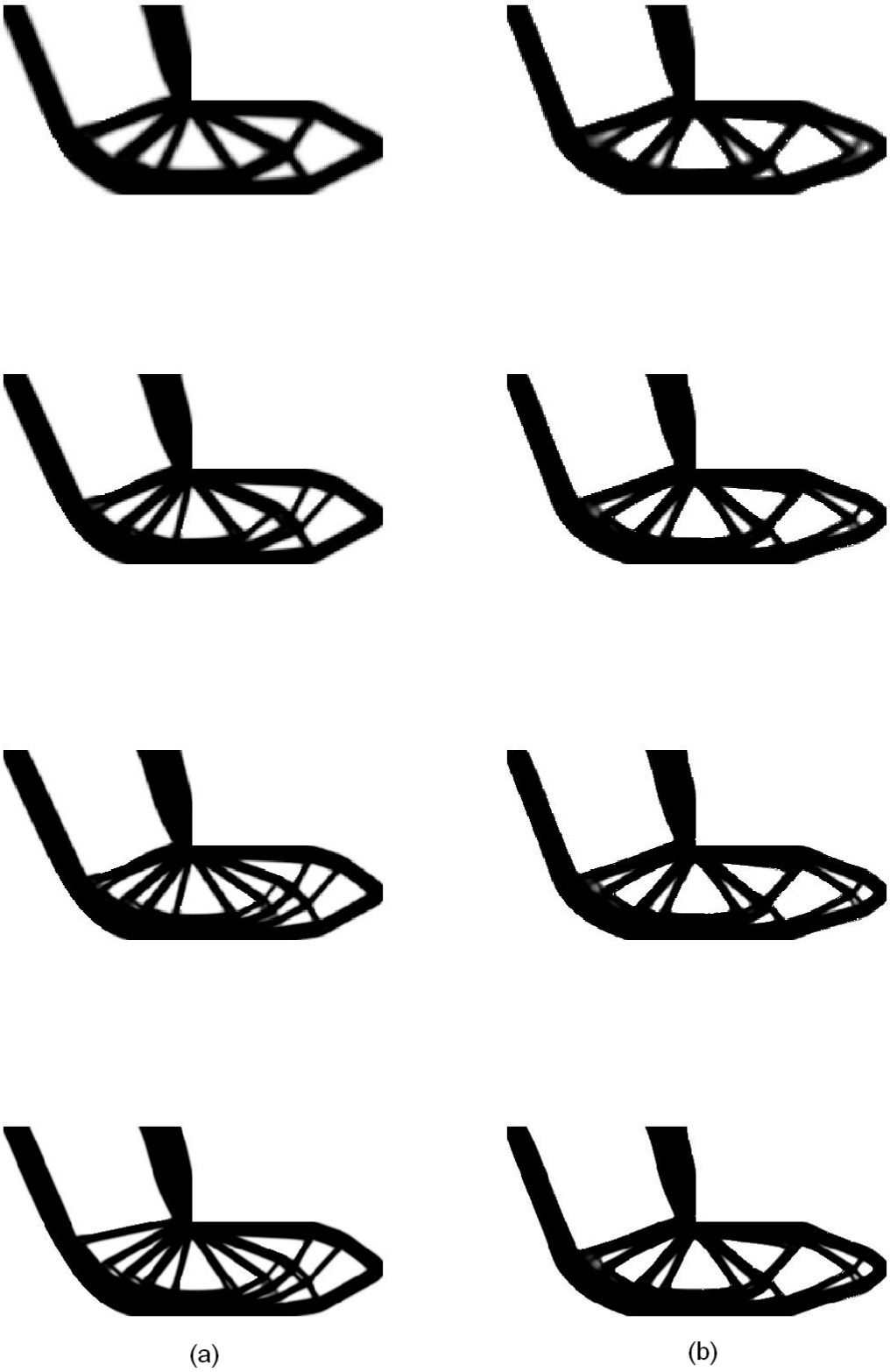


Fig. 5.5 Optimized domain for Test-Example B for each discretization: (a) SIMP output, (b) DL-SCALE output.

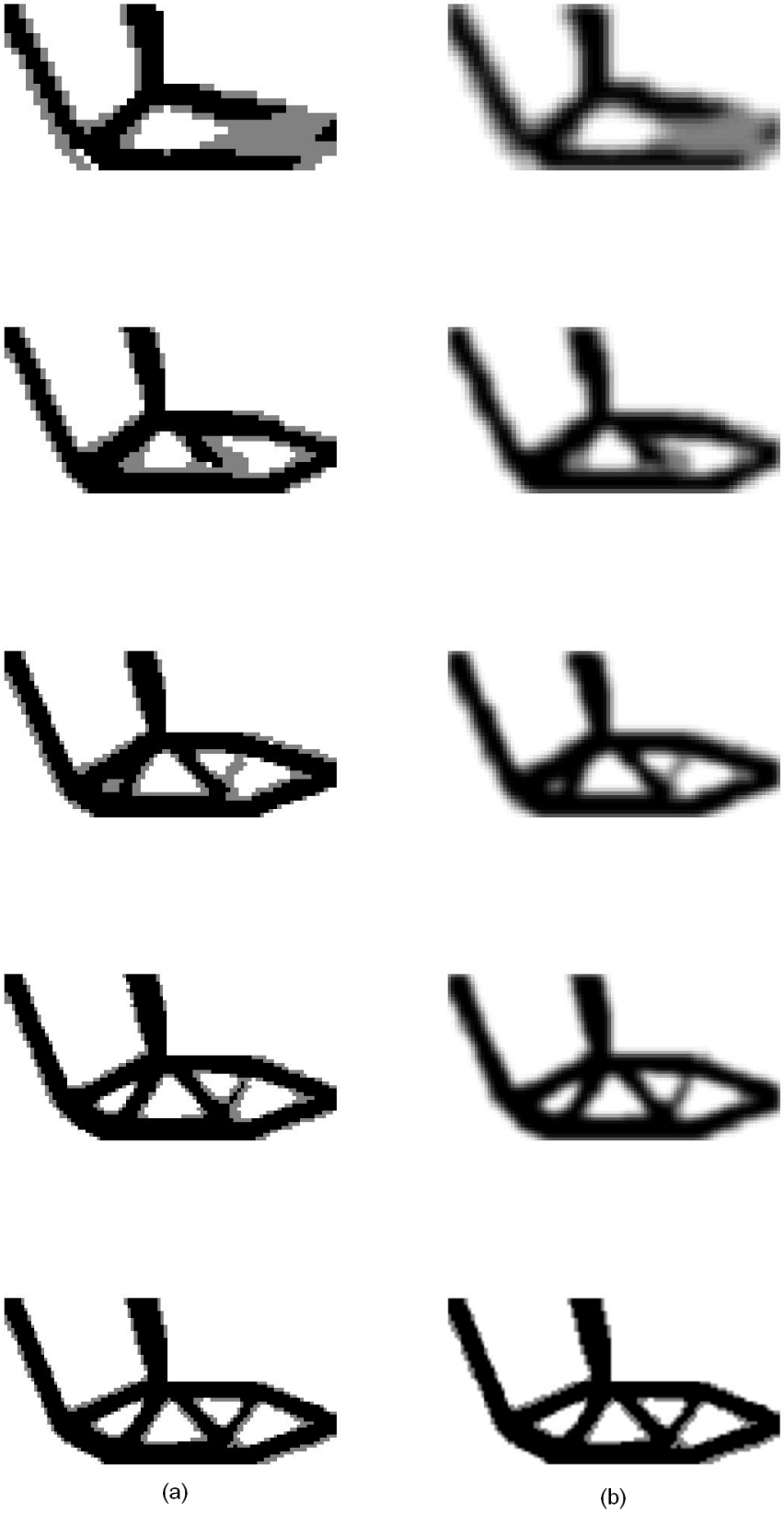


Fig. 5.6 Optimized domains for each discretization of Test-Example B as exported from: (a) DBN, (b) Convolution.

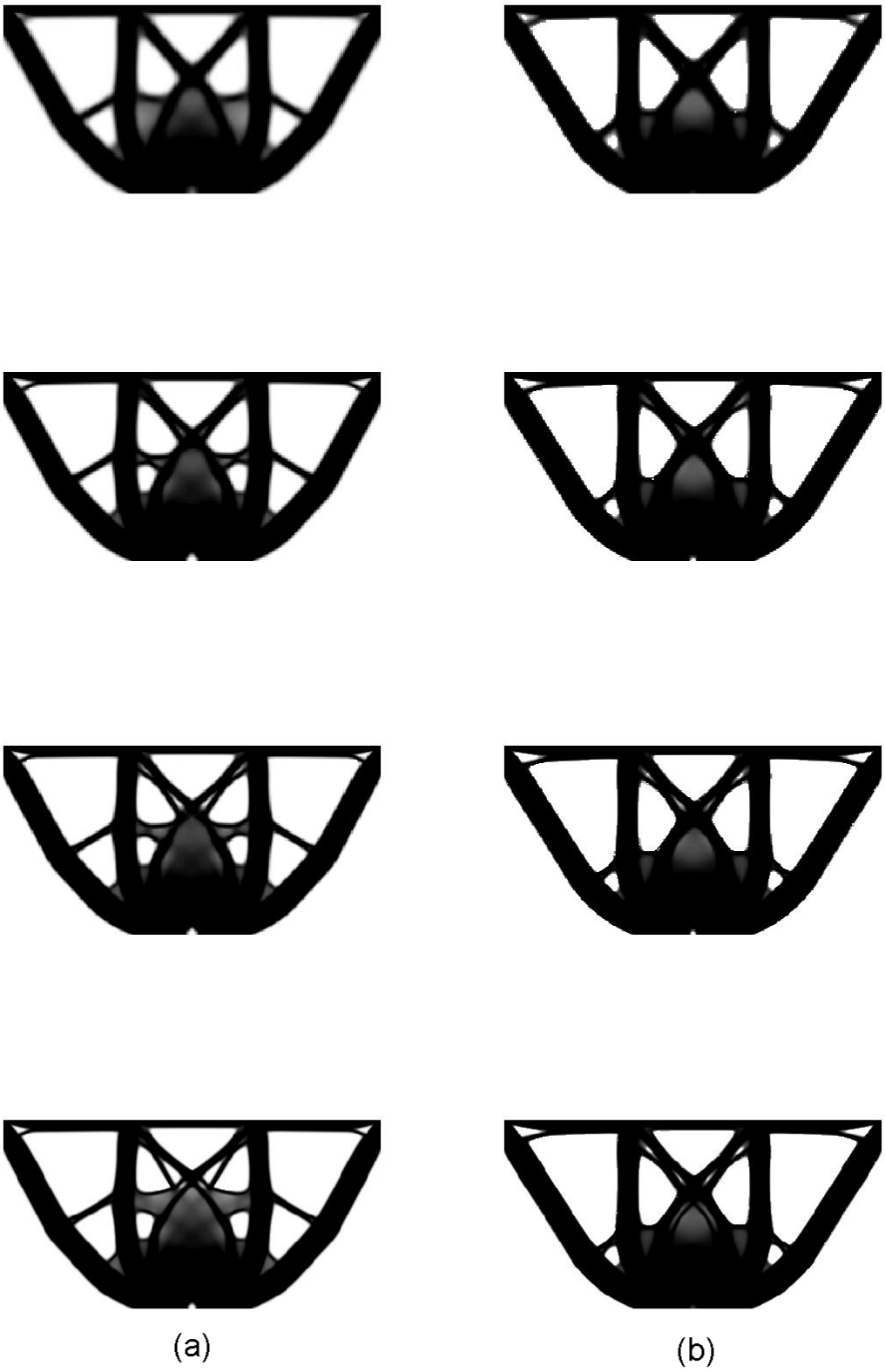


Fig. 5.7 Optimized domain for Test-Example C for each discretization: (a) SIMP output, (b) DL-SCALE output.

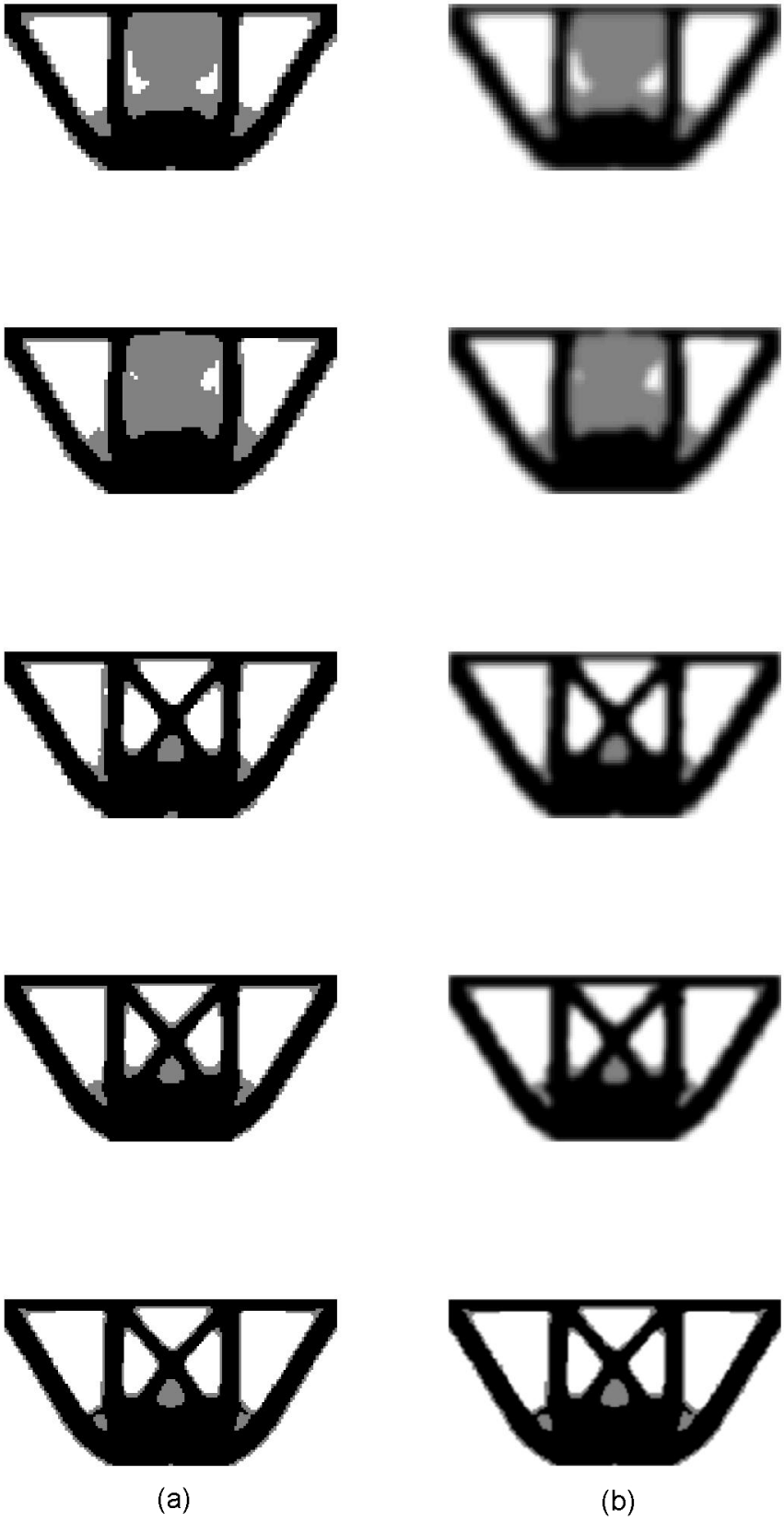


Fig. 5.8 Optimized domains for each discretization of Test-Example C as exported from: (a) DBN, (b) Convolution.

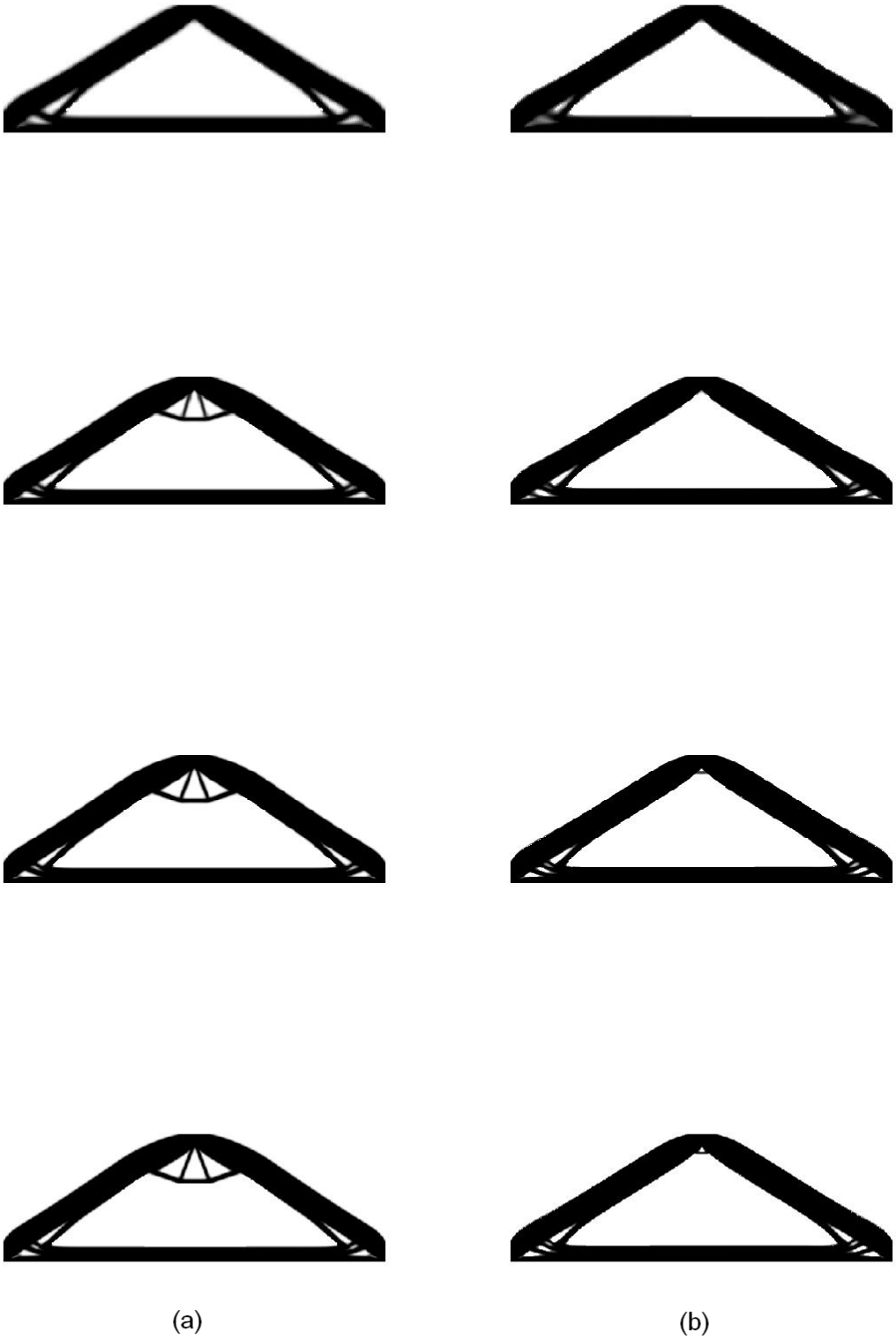


Fig. 5.9 Optimized domain for Test-Example D for each discretization: (a) SIMP output, (b) DL-SCALE output.

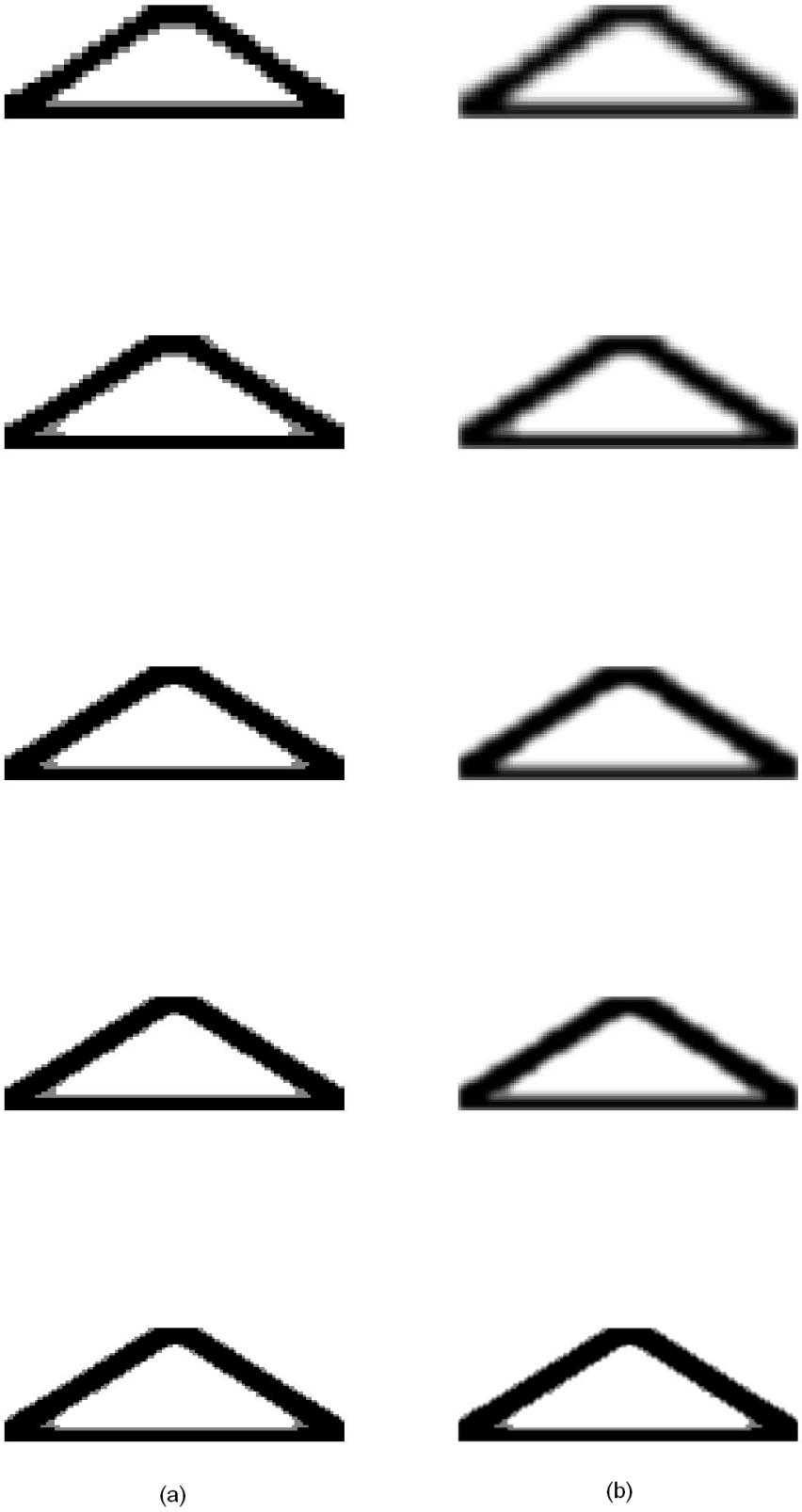


Fig. 5.10 Optimized domains for each discretization of Test-Example D as exported from: (a) DBN, (b) Convolution.

In Test-Example E, DL-SCALE achieved a maximum reduction of computational time equal to 78.04% in the case of 100,000 elements in TOP_f with respect to SIMP while the objective function value was increased by 2.14%. All data recorded in this test can be seen in Table 5.6. Additionally, the final topologies produced by SIMP alone and the ones proposed by DL-SCALE, per discretization can be seen in Fig 5.11(a) and Fig. 5.11(b) respectively. The shapes produced by the DBN for each discretization and the ones from the convolution stage per discretization can be seen in Fig. 5.12(a) and Fig. 5.12(b) respectively.

Table 5.6 DL-SCALE performance in Test-Example E.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
20,000	186	20.88	59.61	58	20.67	40.45	32.15	0.97
50,000	205	20.95	165.86	59	21.21	72.14	56.50	-1.26
75,000	306	21.04	382.83	57	21.44	97.53	74.52	-1.94
100,000	321	21.17	569.66	56	21.62	125.12	78.04	-2.14

5.2 DLRM-TOP - Deep Learning Reduced Order Model Upgrading

In the current section of the dissertation, a reduced order methodology, utilizing a deep learning approach, is proposed for reducing computational load and execution time of topology optimization problems. Reduced order modeling (ROM) or surrogate modeling, is a technique used for downscaling a complex model to a simpler and smaller one while preserving the behavioral aspects of the complex model, targeting to reduced computational needs. The proposed methodology, DLRM-TOP, was inspired based on observation of the performance and behaviour of DL-TOP and DL-SCALE. Both these methods present a different approach to Deep Learning-based, feature learning and prediction in Topology Optimization problems. As previously stated, TO problems are severely greedy in computational demands while this is more evident as population of finite elements in the mesh of the domain to be optimized, increases. For this reason, exploitation of results of simplified models can be very helpful when dealing with a complex one. Final optimized topologies of simple models can provide information for determining the final optimized topology of a the complex one. The proposed methodology is based on this principle while it focuses on reducing computational load as well.

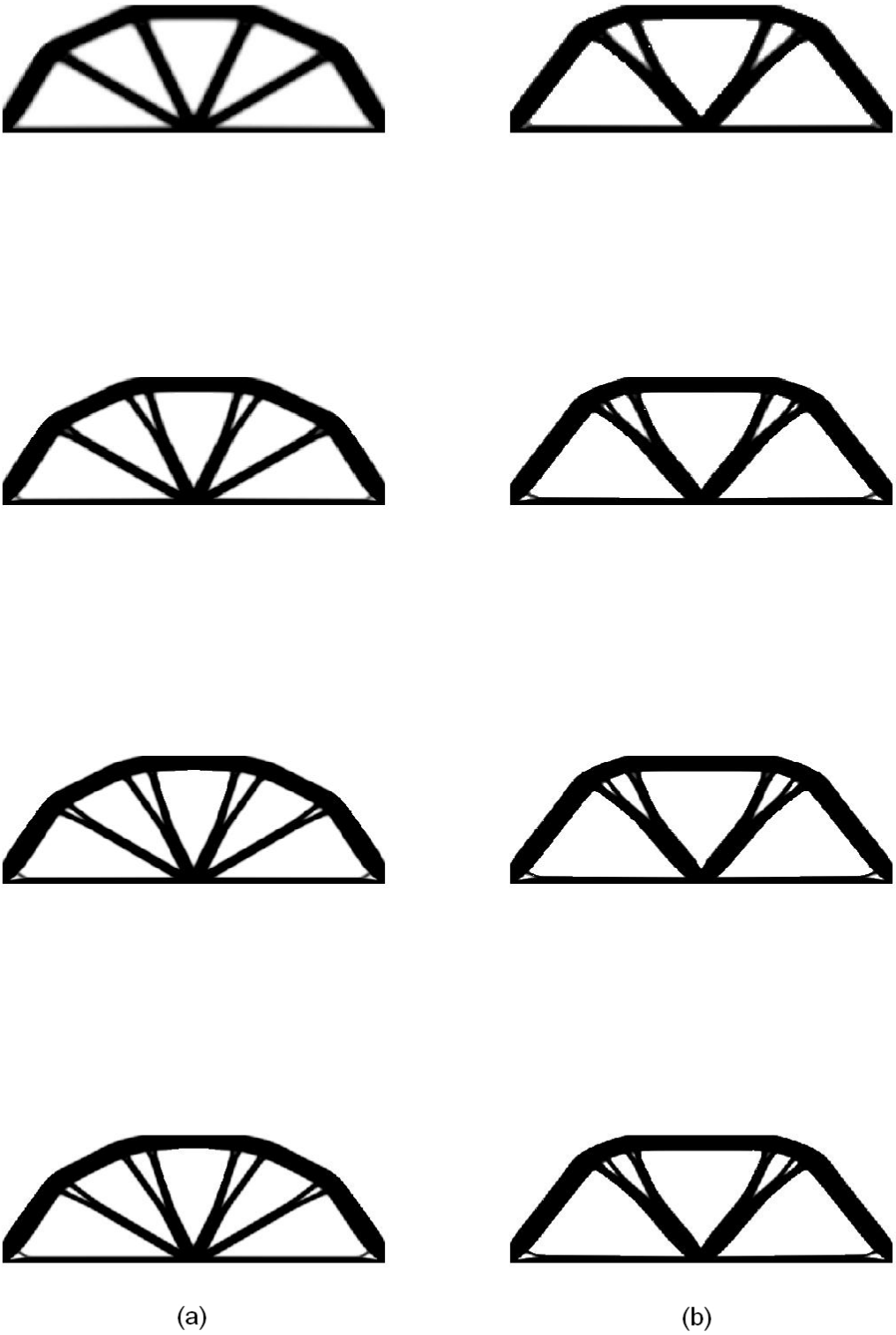


Fig. 5.11 Optimized domain for Test-Example E for each discretization: (a) SIMP output, (b) DL-SCALE output.

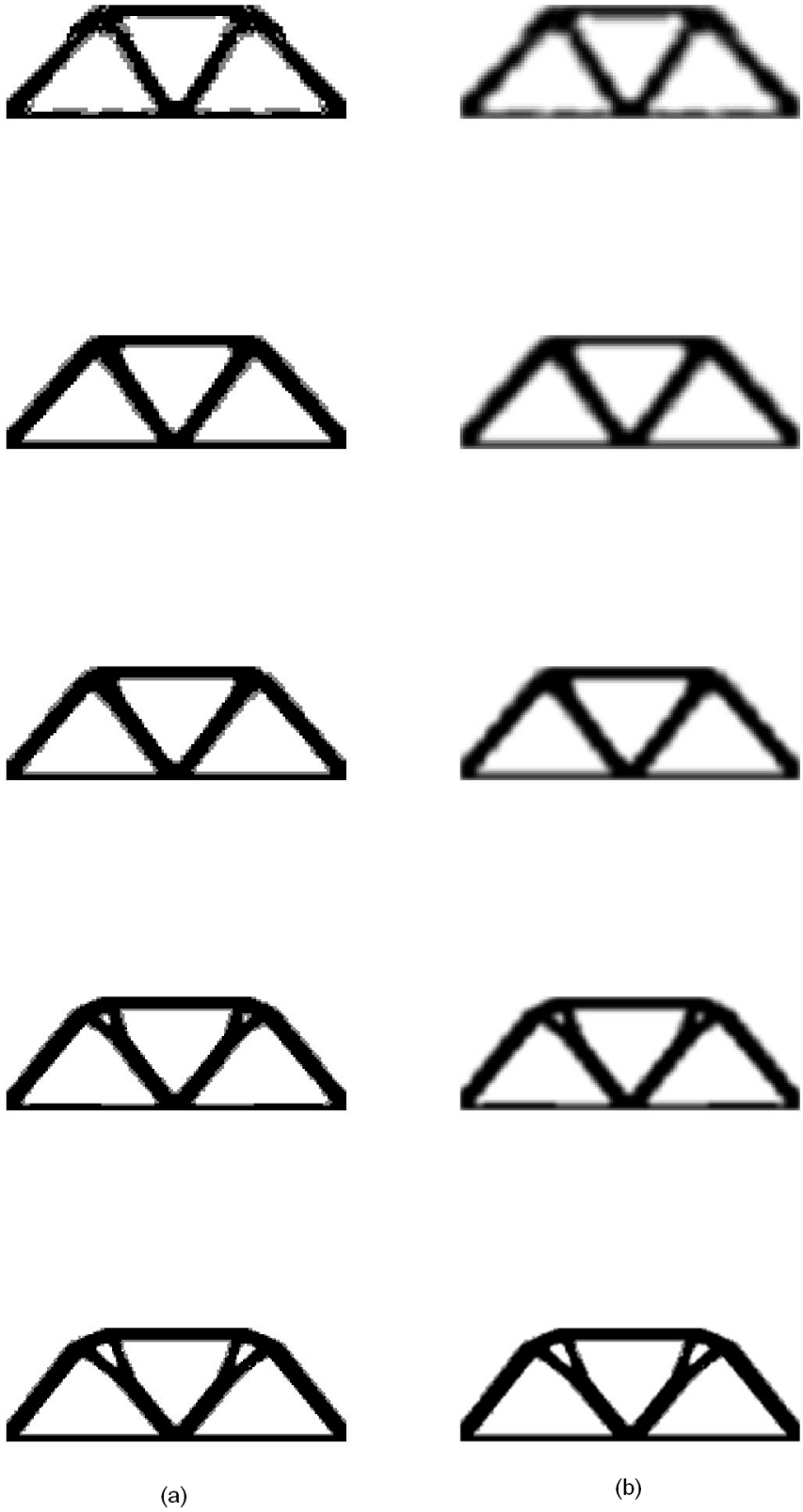


Fig. 5.12 Optimized domains for each discretization of Test-Example E as exported from: (a) DBN, (b) Convolution.

5.2.1 DLRM-TOP methodology description

As described in the DL-SCALE presentation section, an STO problem TOP_f is defined by the model properties $[L^f_{xyz},\ ne^f,\ S^f_c,\ P^f_c,\ V^f_t]$ (domain dimensions, FE population, support conditions, loading conditions and target volume), where $ne^f = ne^f_x * ne^f_y * ne^f_z$. Reduced order STO models with respect to decreased FE population(ne) and identical remaining properties can be described as: $TOP^R_{(i)}:\ [L^f_{xyz},\ ne^{(i)}_R,\ S^f_c\ P^f_c,\ V^f_t]$.
The core procedures of DLRM-TOP methodology are described as follows. Assuming that STO is to be performed in TOP_f, the first necessary step involves generation of five ROMs, $(TOP^R_{(i)})$, with the only difference between each one of them and with TOP_f being the population of FEs. The population of FEs follows the rule described in Eq. 5.1. The $D_{(i)}$ domains for each of the five ROMs are formulated and SIMP is applied to each one of them separately. Contrary to DL-SCALE, SIMP is applied without premature stopping based on initial iterations criterion. The procedure finishes for each ROM once SIMP convergence is achieved and the termination criterion (percentage of change) is satisfied. As a result, the optimized domain $D^{OPT}_{(i)}$ is available. Assuming that $TOP^R_{(i)}$ is a 2D domain, its optimized domain $D^{OPT}_{(i)}$ can be described as follows:

$$D^{OPT}_{(i)} = \begin{bmatrix} d^{opt}_{1,1} & d^{opt}_{1,2} & \dots & d^{opt}_{1,ne^{(i)}_x} \\ d^{opt}_{2,1} & d^{opt}_{2,2} & \dots & d^{opt}_{2,ne^{(i)}_x} \\ \vdots & \vdots & \ddots & \vdots \\ d^{opt}_{ne^{(i)}_y,1} & d^{opt}_{ne^{(i)}_y,2} & \dots & d^{opt}_{ne^{(i)}_y,ne^{(i)}_x} \end{bmatrix} \tag{5.4}$$

In the next step, the initial D_f of TOP_f is created where:

$$D^{(0)}_f = \begin{bmatrix} d^{(0)}_{1,1} & d^{(0)}_{1,2} & \dots & d^{(0)}_{1,ne^{(f)}_x} \\ d^{(0)}_{2,1} & d^{(0)}_{2,2} & \dots & d^{(0)}_{2,ne^{(f)}_x} \\ \vdots & \vdots & \ddots & \vdots \\ d^{(0)}_{ne^{(f)}_y,1} & d^{(0)}_{ne^{(f)}_y,2} & \dots & d^{(0)}_{ne^{(f)}_y,ne^{(f)}_x} \end{bmatrix} \tag{5.5}$$

In the next step, each of the $D^{OPT}_{(i)}$ is used for generating a topology for D_f through interpolation from $ne^{(i)}_R$ to ne_f. As a result, regarding TOP_f, a dataset containing five topologies is

acquired. This dataset, D_{TOP}, can be represented as:

$$D_{TOP} = \begin{bmatrix} T^{(d)}_{1,1} & T^{(d)}_{1,2} & \dots & T^{(d)}_{1,ne^{(f)}_x} \\ T^{(d)}_{2,1} & T^{(d)}_{2,2} & \dots & T^{(d)}_{2,ne^{(f)}_x} \\ \vdots & \vdots & \ddots & \vdots \\ T^{(d)}_{ne^{(f)}_y,1} & T^{(d)}_{ne^{(f)}_y,2} & \dots & T^{(d)}_{ne^{(f)}_y,ne^{(f)}_x} \end{bmatrix} \tag{5.6}$$

$$where \quad T^{(d)}_{i,j} = \left[\{d_{k,l}\}^{opt}_{TOP_1}, \{d_{m,n}\}^{opt}_{TOP_2}, \dots, \{d_{b,c}\}^{opt}_{TOP_5} \right], \; and$$

$$i \in [1, ne^f_y],\; j \in [1, ne^f_x],\; k \in [1, ne^1_y],\; l \in [1, ne^1_x], \dots, b \in [1, ne^5_y],\; c \in [1, ne^5_x]$$

As it can be seen in Eq. 5.6, a population of ne^f vectors $T^{(d)}_{i,j}$ are created which vectors contain the density values of the element of each $D^{OPT}_{(i)}$ that is closer, in terms of geometric centers distance, to the element of TOP_f that $T^{(d)}_{i,j}$ describes.
In the following step of DLRM-TOP, a DBN is used for predicting a final density value for each of the ne^f FEs of TOP_f, based on its $T^{(d)}$. The output of the DBN is used as SIMP input and in the final stage, SIMP performs the necessary iterations until achieving convergence. A flowchart of DLRM-TOP methodology is presented in Fig. 5.13
In order for DBN to be able to discover a correlation between $T^{(d)}$ of each FE and its final density value in $TOP^{(OPT)}_f$, a calibration procedure needs to be performed after first creating a training dataset.

5.2.2 DBN calibration - Training dataset

The training dataset is created by using the two problems presented previously in 4.2.2, the cantilever beam and the simply supported beam. The two examples can be viewed in Fig.

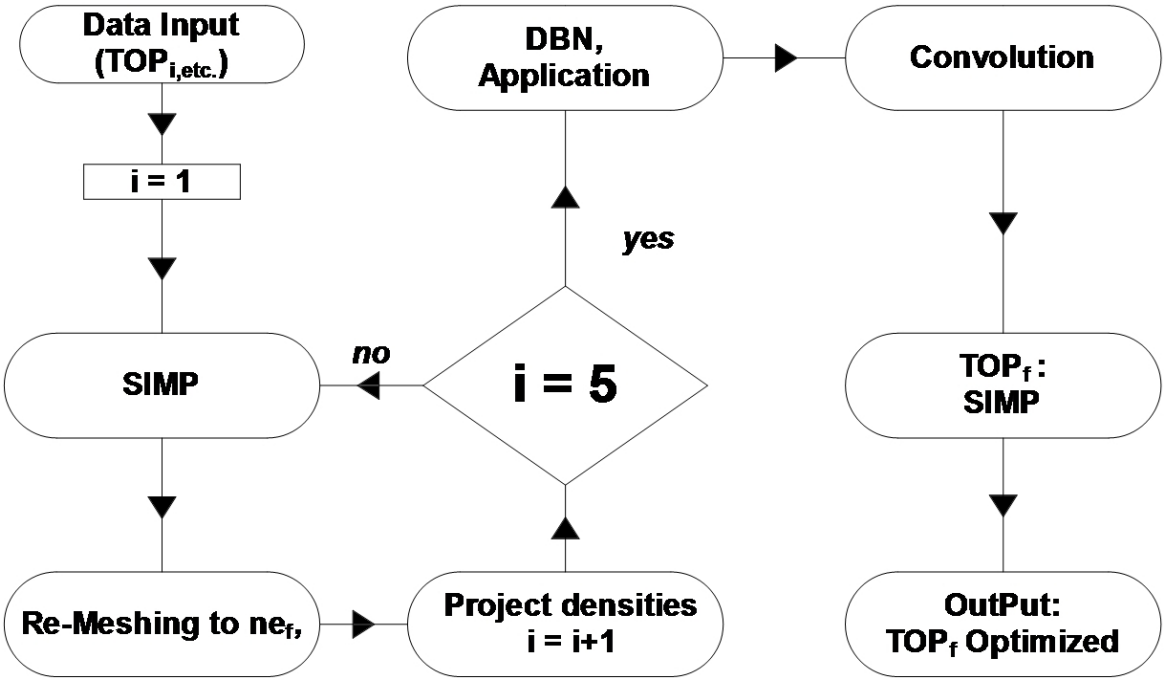


Fig. 5.13 Flowchart of DLRM-TOP methodology

4.5. The required form of the dataset in described in Eq. 5.7.

$$
\begin{array}{cccc}
\left[d^{opt}_{TOP_1}, d^{opt}_{TOP_2}, d^{opt}_{TOP_3}, d^{opt}_{TOP_4}, d^{opt}_{TOP_5}\right]^{(1)} & & \rightarrow & d_{1,TOP_f} \\
\left[d^{opt}_{TOP_1}, d^{opt}_{TOP_2}, d^{opt}_{TOP_3}, d^{opt}_{TOP_4}, d^{opt}_{TOP_5}\right]^{(2)} & & \rightarrow & d_{2,TOP_f} \\
\vdots \qquad \vdots & \ddots & \vdots & \\
\left[d^{opt}_{TOP_1}, d^{opt}_{TOP_2}, d^{opt}_{TOP_3}, d^{opt}_{TOP_4}, d^{opt}_{TOP_5}\right]^{(ne-1)} & & \rightarrow & d_{ne-1,TOP_f} \\
\left[d^{opt}_{TOP_1}, d^{opt}_{TOP_2}, d^{opt}_{TOP_3}, d^{opt}_{TOP_4}, d^{opt}_{TOP_5}\right]^{(ne)} & & \rightarrow & d_{ne,TOP_f} \\
Input & & & Target
\end{array}
\tag{5.7}
$$

For each of the test examples, the population of FEs of TOP_f is chosen equal to 200,000. Additionally, 5 different combinations of $TOP^R_{(i)}$ are used while the population of FEs of

each TOP^R in each combination is presented in Eq. 5.8 for test example i:

$$DAT_i = \begin{bmatrix} 1,000 & 2,000 & 3,000 & 4,000 & 5,000 \\ 2,000 & 3,000 & 4,000 & 5,000 & 6,000 \\ 4,000 & 5,000 & 6,000 & 7,000 & 8,000 \\ 2,000 & 5,000 & 7,000 & 8,000 & 10,000 \\ 6,000 & 7,000 & 8,000 & 9,000 & 10,000 \end{bmatrix} \tag{5.8}$$

Both DAT_1 and DAT_2 are joined and the training dataset is formulated, consisting of nearly 1,500,000 cases of $[T_{i,j}^{(d)} \rightarrow d_i^{TOP_f}]$. Once calibration is completed, DLRM-TOP can be applied to any STO problem without the need to recalibrate as density fluctuations of any FE with respect to model upgrading is examined. The performance of DLRM-TOP is evaluated against five test examples as presented in the next section.

5.2.3 DLRM-TOP performance

The performance evaluation of DLRM-TOP methodology, is based on a series of five 2D test-examples, known to literature. The test-cases are solved using SIMP methodology and record is kept regarding necessary iterations for convergence, objective function value (compliance) and execution time. The same test-examples are also optimized with the use of DLRM-TOP methodology and the same records are kept. As according to DLRM-TOP, iterations are performed on domains with less dense meshes, the comparison is based solely on execution time while iterations are mentioned just as additional information. In the case of SIMP, the total time from the domain definition and up to the end of the final iteration is recorder. In the case of DLRM-TOP, the time from the definition of $D_{(i)}^{OPT}$ to the end of the final iteration of $D_f^{(0)}$ is recorded. As DLRM-TOP requires the formulation of five domains with reduced population of FEs per test case for creating $D_{(i)}^{OPT}$, the population of $ne^{(i)}$s is equal to [5000, 7000, 10000, 15000, 20000] respectively. Regarding the population of FEs in the final domain of each test-example, four different populations are examined in order to evaluate the performance of DLRM-TOP. These four populations are: $[75,000,\ 100,000,\ 150,000,\ 200,000]$. The five test-examples used are described bellow.

Test-Examples description

In Test-Example A, the support conditions refer to fully fixed boundary conditions placed along the y axis at the left side of the domain and the loading condition refers to two concentrated forces P along the y axis and applied at the lower and upper corner of the right

side of the domain. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 30%.

In Test-Example B, the support conditions refer to fully fixed boundary condition along the x axis ending at the half of the x dimension and the loading condition refers to one concentrated force P along the y axis and applied in the fourth of the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 30%.

In Test-Example C, the support conditions refer to fully fixed boundary conditions along the x axis, starting at the 3:8ths of the x dimension and the loading conditions refer to four concentrated forces P along the y axis and applied at intermediate distances equal to 1:3 of the x dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 45%.

In Test-Example D, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y axis, applied in the middle of the y dimension at the base of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 30%.

In Test-Example E, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y, applied at the half of the y dimension at the top of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 30%.

It is worth pointing out that a sensitivity filter with radius equal to 3 was chosen in all cases as SIMP filter. A schematic representation of the test-examples can be seen in Fig. 5.23.

5.2.4 Results

In Test-Example A, DLRM-TOP achieved a maximum reduction of computational time equal to 80.64% in the case of 200,000 elements in $D_f^{(0)}$ with respect to SIMP while the objective function value was increased by 0.73%. All data recorded in this test can be seen in Table 5.7. Additionally, the final topologies produced by SIMP alone and the ones proposed by DLRM-TOP, per discretization can be seen in Fig 5.15(a) and Fig. 5.15(b) respectively.

Table 5.7 DLRM-TOP performance in Test-Example A.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
75,000	251	248.41	404.89	80	249.40	219.40	45.87	-0.40
100,000	331	246.46	736.53	61	247.37	224.06	69.58	-0.37
150,000	340	243.52	1203.95	72	245.15	337.66	71.95	-0.67
200,000	521	242.62	2559.05	86	244.39	495.55	80.64	-0.73

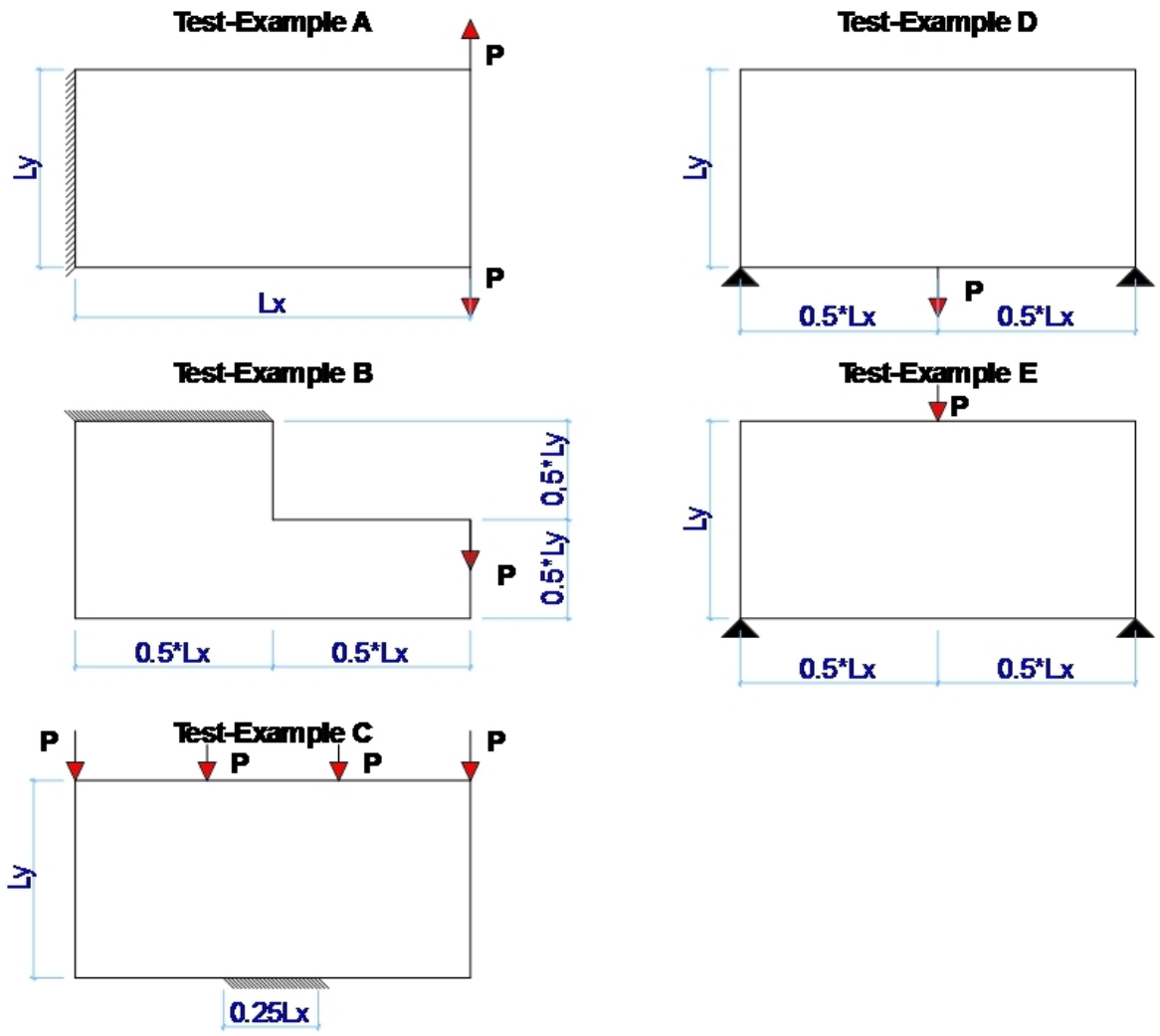


Fig. 5.14 Schematic representation of Test-Examples

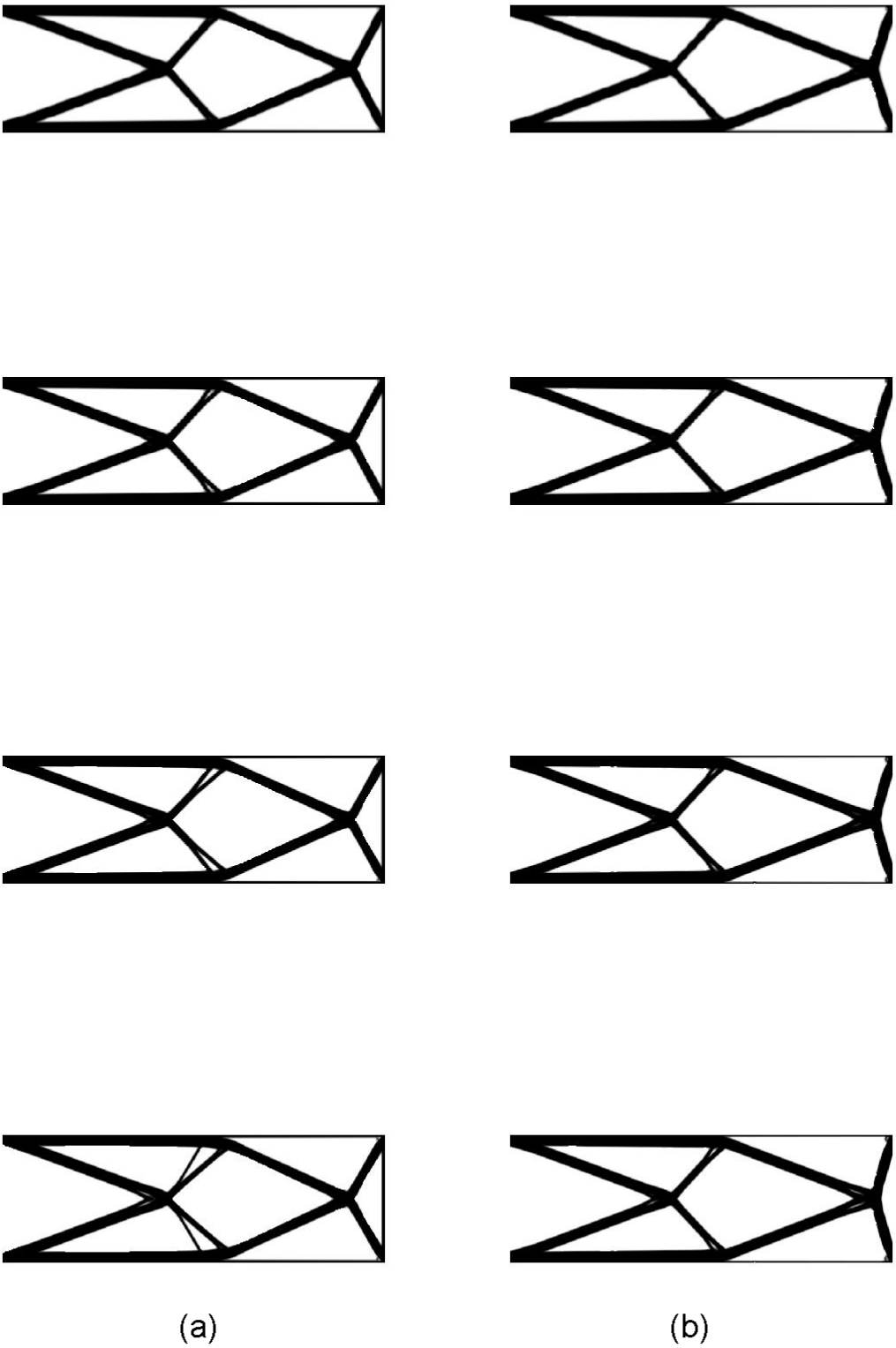


Fig. 5.15 Optimized domain for Test-Example A for each discretization: (a) SIMP output, (b) DL-SCALE output.

In Test-Example B, DLRM-TOP achieved a reduction of computational time equal to 56.91% in the case of 200,000 elements in $D_f^{(0)}$ with respect to SIMP while the objective function value was decreased by 0.58%. All data recorded in this test can be seen in Table 5.8. Additionally, the final topologies produced by SIMP alone and the ones proposed by DLRM-TOP, per discretization can be seen in Fig 5.16(a) and Fig. 5.16(b) respectively.

Table 5.8 DLRM-TOP performance in Test-Example B.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
75,000	401	159.86	505.88	79	158.83	292.43	42.19	0.64
100,000	444	158.44	774.02	58	157.83	291.76	62.31	0.38
150,000	514	156.35	1438.26	98	155.73	461.50	67.91	0.40
200,000	367	156.01	1382.55	107	155.10	595.76	56.91	0.58

In Test-Example C, DLRM-TOP achieved a reduction of computational time equal to 78.69% in the case of 200,000 elements in $D_f^{(0)}$ with respect to SIMP while the objective function value was increased by 0.11%. All data recorded in this test can be seen in Table 5.9. Additionally, the final topologies produced by SIMP alone and the ones proposed by DLRM-TOP, per discretization can be seen in Fig 5.17(a) and Fig. 5.17(b) respectively.

Table 5.9 DLRM-TOP performance in Test-Example C.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
75,000	131	109.84	225.78	61	109.73	196.48	12.98	0.10
100,000	438	111.53	1025.44	65	111.03	243.43	76.26	0.45
150,000	509	111.52	1888.79	74	21.62	321.29	61.44	0.09
200,000	556	112.51	2838.85	101	112.63	604.94	78.69	-0.11

In Test-Example D, DLRM-TOP achieved a reduction of computational time equal to 60.58% in the case of 200,000 elements in $D_f^{(0)}$ with respect to SIMP while the objective function value was increased by 0.01%. All data recorded in this test can be seen in Table 5.10. Additionally, the final topologies produced by SIMP alone and the ones proposed by DLRM-TOP, per discretization can be seen in Fig 5.18(a) and Fig. 5.18(b) respectively.

Table 5.10 DLRM-TOP performance in Test-Example D.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
75,000	261	21.30	330.54	74	21.22	204.30	38.19	0.38
100,000	326	21.45	559.69	79	21.38	252.59	54.87	0.33
150,000	310	21.60	833.30	74	21.62	321.29	61.44	-0.06
200,000	358	21.79	1306.80	99	21.79	515.14	60.58	-0.01

In Test-Example E, DLRM-TOP achieved a reduction of computational time equal to 67.48% in the case of 200,000 elements in $D_f^{(0)}$ with respect to SIMP while the objective

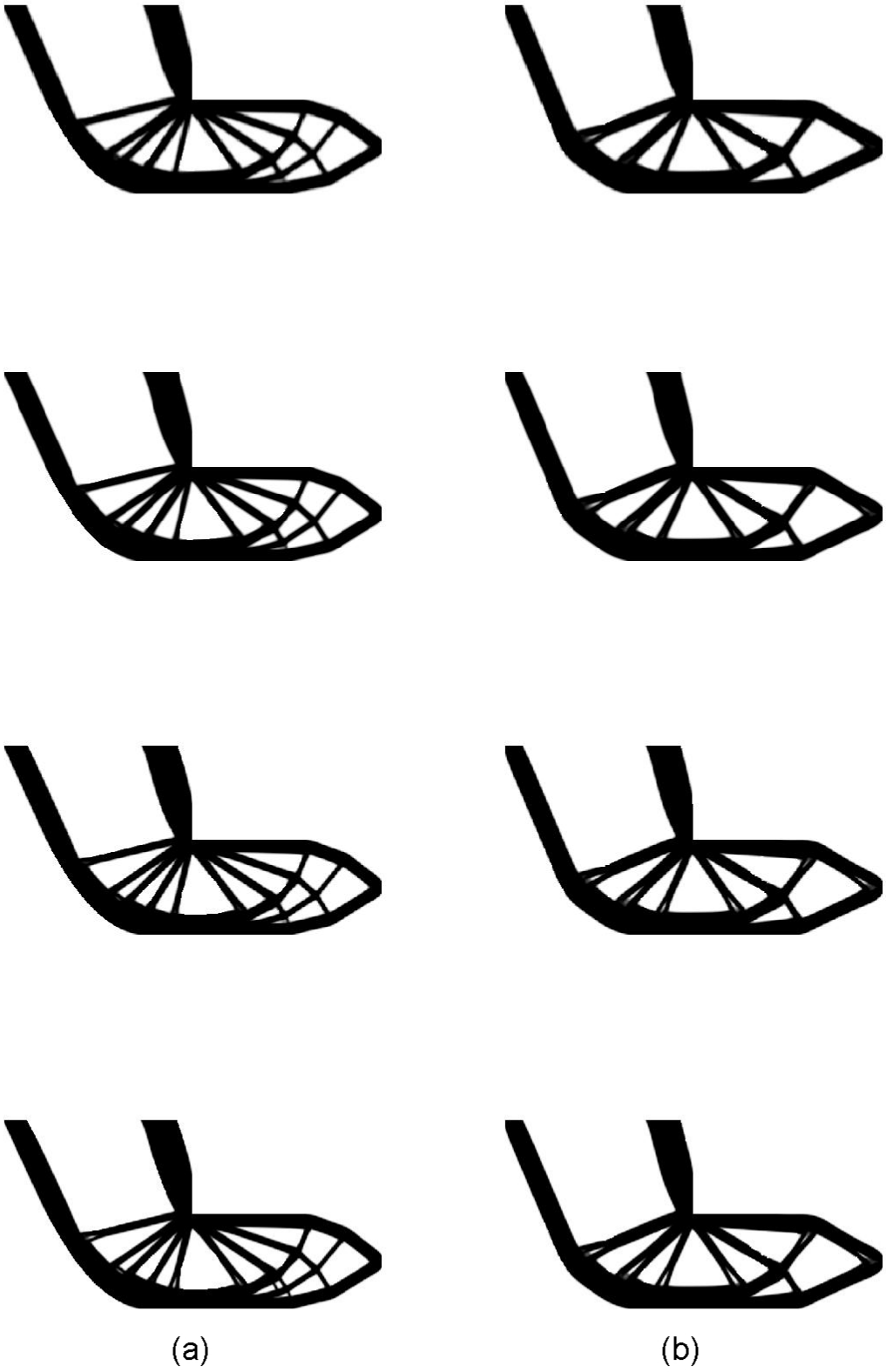


Fig. 5.16 Optimized domain for Test-Example B for each discretization: (a) SIMP output, (b) DL-SCALE output.

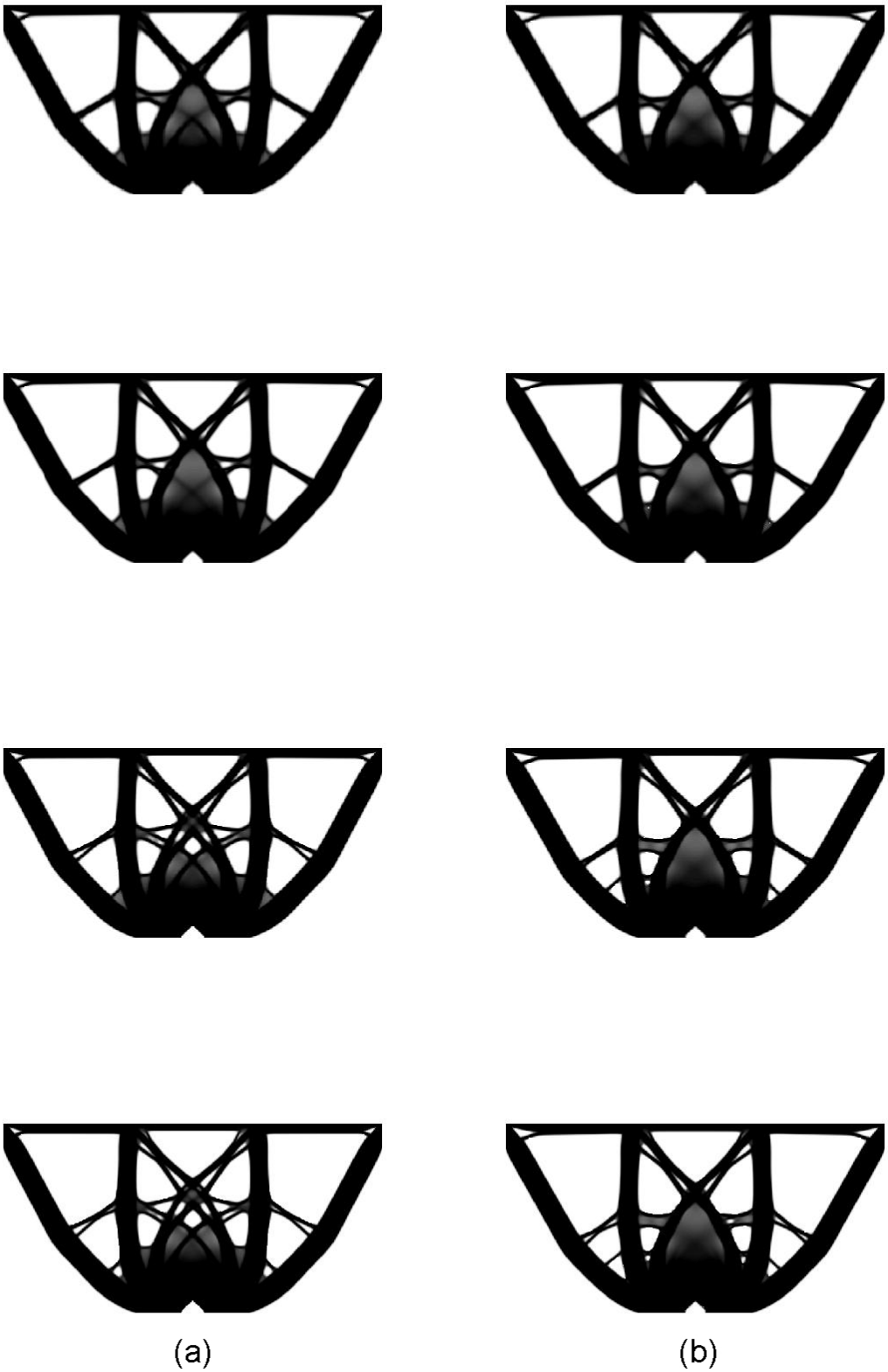


Fig. 5.17 Optimized domain for Test-Example C for each discretization: (a) SIMP output, (b) DLRM-TOP output.

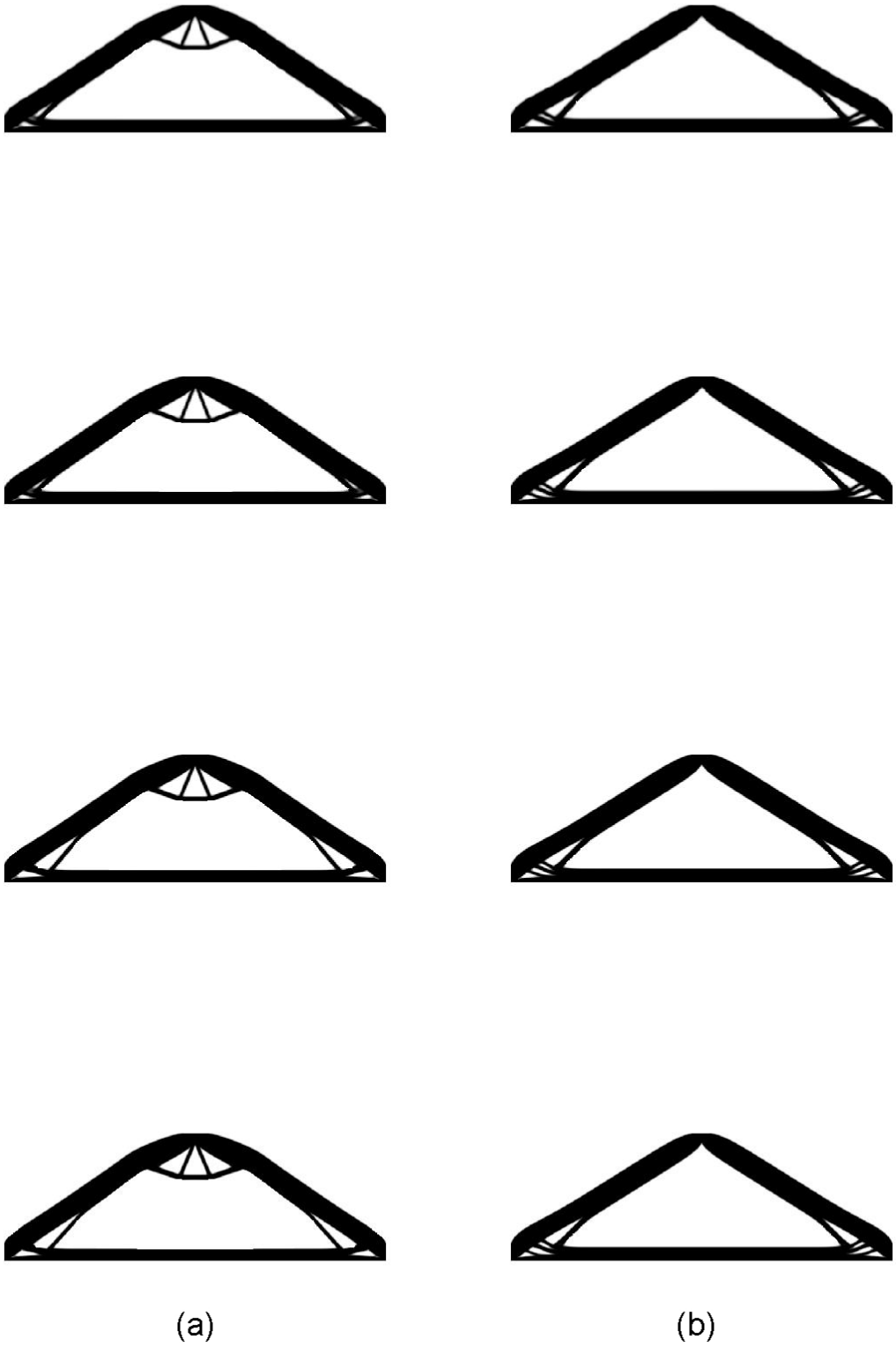


Fig. 5.18 Optimized domain for Test-Example D for each discretization: (a) SIMP output, (b) DLRM-TOP output.

function value was decreased by 0.77%. All data recorded in this test can be seen in Table 5.11. Additionally, the final topologies produced by SIMP alone and the ones proposed by DLRM-TOP, per discretization can be seen in Fig 5.19(a) and Fig. 5.19(b) respectively.

Table 5.11 DLRM-TOP performance in Test-Example E.

ne	SIMP			DL-SCALE			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
75,000	314	23.71	462.43	80	23.72	246.44	46.71	-0.05
100,000	393	23.79	813.58	114	23.78	261.53	55.56	0.08
150,000	429	23.99	1396.71	183	23.87	718.71	48.54	0.49
200,000	573	24.16	2538.61	156	23.97	820.53	67.68	0.77

5.3 CN-TOP - Deep Learning Model Enhancing

In recent bibliography, a lot of work is presented on improving image quality and upscaling image or even video resolution [43, 85, 123, 154, 223] with the use of deep neural networks (mainly generative adversarial networks and convolutional neural networks). These approaches are being used in several forms from object recognition to security identification applications. Convolutional neural networks have also been applied in TO problems in a different manner previously [198]. In the current section of the dissertation, the application of such methodology for reducing computational load of TO in a reduced order model manner.

5.3.1 CN-TOP methodology description

As previously stated, TO result of a 2D problem is a matrix D_v where each value represents the material density in the current FE. For example, an optimized domain as described in Eq.

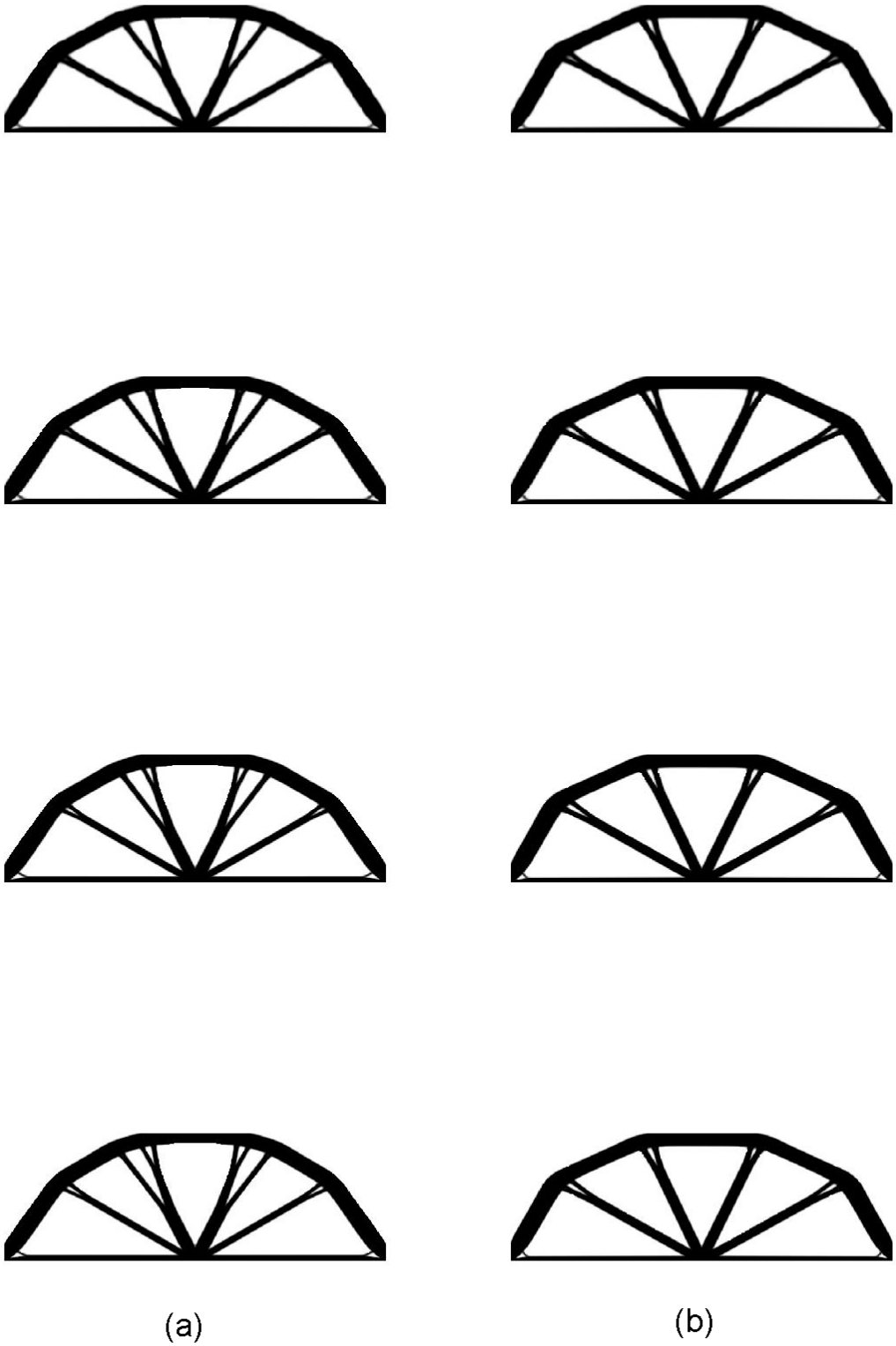


Fig. 5.19 Optimized domain for Test-Example E for each discretization: (a) SIMP output, (b) DLRM-TOP output.

5.9 is visualized in gray-scale as it can be seen in Fig. 5.20.

$$D_v = \begin{bmatrix} 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \\ 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 0.86 & 0.81 & 0.80 & 0.79 & 0.79 & 0.79 \\ 0.27 & 0.37 & 0.57 & 0.75 & 0.81 & 0.68 & 0.46 & 0.28 & 0.21 & 0.20 & 0.21 \\ 0.01 & 0.03 & 0.12 & 0.34 & 0.62 & 0.78 & 0.66 & 0.38 & 0.15 & 0.05 & 0.02 \\ 0.00 & 0.00 & 0.01 & 0.09 & 0.29 & 0.57 & 0.78 & 0.73 & 0.49 & 0.25 & 0.13 \\ 0.00 & 0.00 & 0.00 & 0.01 & 0.05 & 0.20 & 0.47 & 0.71 & 0.76 & 0.59 & 0.42 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 0.13 & 0.36 & 0.62 & 0.75 & 0.73 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.03 & 0.17 & 0.45 & 0.73 & 0.87 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 0.13 & 0.36 & 0.62 & 0.75 & 0.73 \\ 0.00 & 0.00 & 0.00 & 0.01 & 0.05 & 0.20 & 0.47 & 0.71 & 0.76 & 0.59 & 0.42 \\ 0.00 & 0.00 & 0.01 & 0.09 & 0.29 & 0.57 & 0.78 & 0.73 & 0.49 & 0.25 & 0.13 \\ 0.01 & 0.03 & 0.12 & 0.34 & 0.62 & 0.78 & 0.66 & 0.38 & 0.15 & 0.05 & 0.02 \\ 0.27 & 0.37 & 0.57 & 0.75 & 0.81 & 0.68 & 0.46 & 0.28 & 0.21 & 0.20 & 0.21 \\ 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 0.86 & 0.81 & 0.80 & 0.79 & 0.79 & 0.79 \\ 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 \end{bmatrix} \tag{5.9}$$

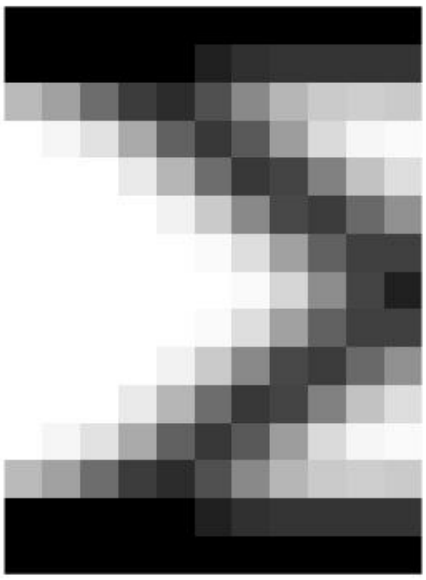


Fig. 5.20 Visualization of optimized domain D_v according to Eq. 5.9.

Every image, can be described as a matrix with $[x, y, z]$ dimensions where x, y define the image resolution (i.e. the number of pixels) and z represents the number of colour channels. By increasing the resolution of an image, magnification of an image is possible with restricted noise per pixels. In the case of TO, this means that information stored per pixel is increased resulting to a smoothed image. This can be translated into increasing the mesh of the 2D domain without adding significant amount of error in the density information for each FE. Increasing image resolution techniques are divided into two major categories:

- Single-frame super-resolution, where only one frame of an image is used for increasing the resolution of the same image
- Multi-frame super-resolution, where several frames of the image are used for generating a super-resolution image

with several methods proposed in each of the above categories. As generating frames in TO is time-consuming, the first category is more convenient in this case. One of the most well-known methods for single-frame image super-resolution is the Fast Super-Resolution Convolutional Neural Network (FSRCNN) [43] which is based on Super-Resolution Convolutional Neural Network (SRCNN) [42] firstly introduced in 2014.

FSRCNN

FSRCNN is a deep convolutional neural network developed for performing single-frame image super-resolution. FSRCNN which consists of five functional modules [43]. These modules are:

- Feature extraction (Convolution)
- Downsize (Convolution)
- Mapping (Convolution)
- Upscale (Convolution)
- De-convolution

FSRCNN can be schematically represented as seen in Fig. 5.21.

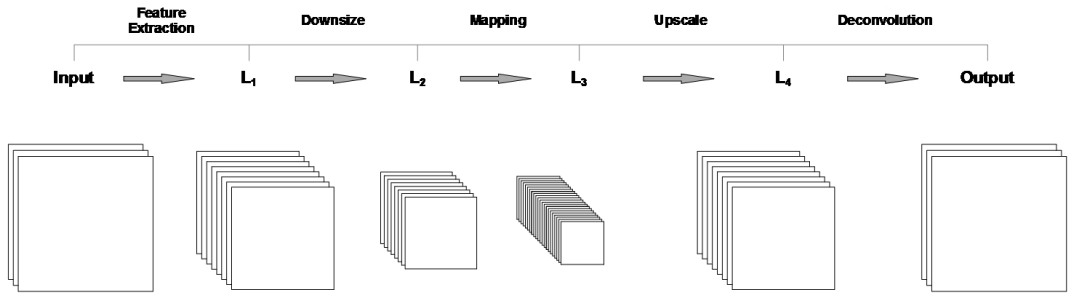


Fig. 5.21 Schematic representation of an FSRCNN.

CN-TOP methodology

As previously described, a 2D optimized domain D_i of size $[n_y, n_x]$ can be represented as a 2D matrix as seen in Eq. 5.10.

$$D_i = \begin{bmatrix} d_{1,1} & d_{1,2} & \dots & d_{1,n_x} \\ d_{2,1} & d_{2,2} & \dots & d_{2,n_x} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n_y,1} & d_{n_y,2} & \dots & d_{n_y,n_x} \end{bmatrix} \tag{5.10}$$

where $d_{i,j}$ is the density value of the $[i, j]$ FE, while the image of this topology can be described by the Im_i matrix presented in Eq. 5.11.

$$Im_i = \begin{bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,p_x} \\ v_{2,1} & v_{2,2} & \dots & v_{2,p_x} \\ \vdots & \vdots & \ddots & \vdots \\ v_{p_y,1} & v_{p_y,2} & \dots & v_{p_y,p_x} \end{bmatrix} \tag{5.11}$$

where $v_{i,j}$ is the pixel colour value of the $[i, j]$ in gray-scale and p_x, p_y is the pixel number per axis. As stated in a previous section, the computational time for applying SIMP is heavily depended on the number of FEs in the domain. The basic idea behind CN-TOP is to optimize a coarse domain with SIMP and use a trained FSRCNN network to increase the resolution of the SIMP result.

The basic steps of CN-TOP are:

- SIMP on coarse domain
- Increased resolution on SIMP output via FSRCNN
- Dense re-meshing of FSRCNN output
- Applying SIMP on re-meshed domain

On the first step, the coarse domain D_c is created with $x_c * y_c$ number of FEs. This domain is optimized with the use of SIMP and an optimal topology D_c^{opt} is acquired. The image of D_c^{opt} is then fed into the trained FSRCNN network and the network outputs a topology image with increased resolution, Im_{SR}^{opt} in the second step. In the third step, the Im_{SR}^{opt} image is translated into a domain D_{SR}^{opt} and this domain is re-meshed in a dense manner, resulting to D_{dense}^{opt} with $a^2 * x_c * y_c$ number of FEs where a is the multiplying factor defining the size proportion between the coarse and the dense domain. The density value of each element in the dense domain is set equal to the one with the minimum geometric distance from the coarse domain. Before the fifth step, a convolution is performed on D_{dense}^{opt} with the use of a dilate smoothing filter (d_{flt}). An example of such a filter can be seen in Eq. 5.12.

$$d_{flt} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.12}$$

In the final step, D_{dense}^{opt} is used as SIMP input and optimization is performed until convergence is achieved and the final topology D_f^{opt} is acquired. A flowchart of CN-TOP can be seen in Fig. 5.22. As it can be seen previously, a trained FSRCNN is needed for applying CN-TOP method. This network needs to be trained in recognizing transformations between TO problems from coarse to dense meshes. In order for this to be possible, a training set needs to be defined.

5.3.2 CNN calibration - Training dataset

The calibration of FSRCNN requires the generation of a training dataset that consists of pairs of input and output optimized topologies where the only difference between member of the same pair is the population of FEs. In detail, a dataset consisting of 10,000 pairs (20,000 topologies) is generated by using the code presented by Sosnovik and Oseledets [198] and Hunter et al. [84]. With the use of these codes, 10,000 pairs of optimized topologies where

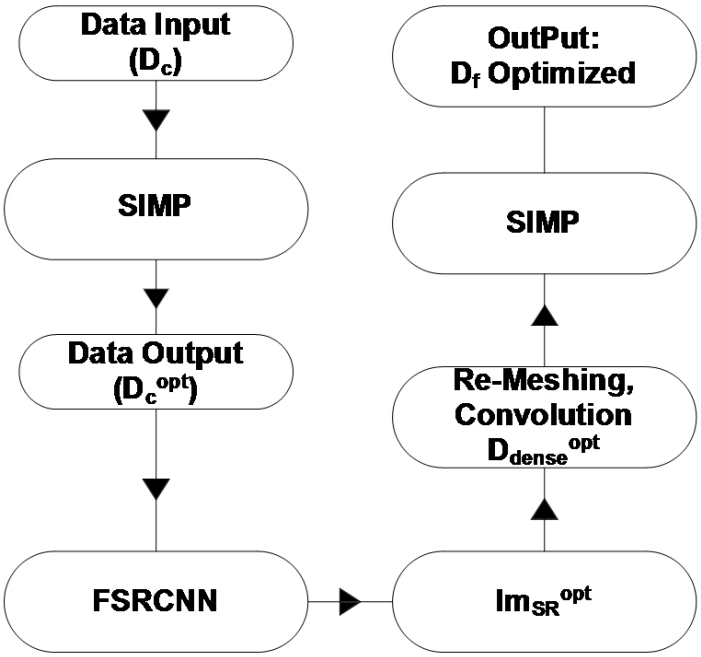


Fig. 5.22 Flowchart of CN-TOP method.

$x_c = y_c = 40$ and $a = 4$ where created. The FSRCNN was trained on the above generated dataset before being used in CN-TOP method.

5.3.3 CN-TOP performance

CN-TOP methodology performance evaluation, is based on five 2D test-examples, known to literature. The test-cases are solved using SIMP methodology and record is kept regarding necessary iterations for convergence, objective function value (compliance) and execution time. The same test-examples are also optimized with the use of CN-TOP methodology and the same records are kept. As according to CN-TOP, iterations are performed on a domain with less dense mesh along with the final one, the comparison is based solely on execution time while iterations are mentioned just as additional information. In the case of SIMP, the total time from the domain definition and up to the end of the final iteration is recorder. In the case of CN-TOP, the time from the definition of D_c to the end of the final iteration of D_f is recorded. As CN-TOP requires the formulation of a domain with reduced population of FEs per test case for creating D_c, the population of ne_c is equal to 20000. Regarding the population of FEs in the final domain, it is chosen equal to $180,000$, nine times bigger than the reduced one ($a = 3$). The five test-examples used are described below.

Test-Examples description

In Test-Example A, the support conditions refer to two fixed joints placed at the two right corners of the domain and the loading conditions refer to two concentrated forces P along the x axis and applied in the left and right middle of the span in the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 30%.

In Test-Example B, the support conditions refer to fully fixed boundary condition along the x axis ending at the half of the x dimension and the loading condition refers to one concentrated force P along the y axis and applied in the fourth of the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 30%.

In Test-Example C, the support conditions refer to fully fixed boundary conditions along the x axis, starting at the 3:8ths of the x dimension and the loading conditions refer to four concentrated forces P along the y axis and applied at intermediate distances equal to 1:3 of the x dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 50%.

In Test-Example D, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y axis, applied in the middle of the y dimension at the base of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 30%.

In Test-Example E, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y, applied at the half of the y dimension at the top of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 30%.

It is worth pointing out that a sensitivity filter with radius equal to 3 was chosen in all cases as SIMP filter. A schematic representation of the test-examples can be seen in Fig. 5.23.

5.3.4 Results

In Test-Example A, CN-TOP achieved a reduction of computational time equal to 67.90% with respect to SIMP while the objective function value was decreased by 0.25%. The final topology produced by SIMP alone, the one proposed by CN-TOP, the CNN proposed topology and the one exported from convolution can be seen in Fig 5.24(a),(b),(c) and Fig. 5.24(d) respectively. In Test-Example B, CN-TOP achieved a reduction of computational time equal to 75.22% with respect to SIMP while the objective function value was decreased by 0.64%. The final topology produced by SIMP alone, the one proposed by CN-TOP, the CNN proposed topology and the one exported from convolution can be seen in Fig

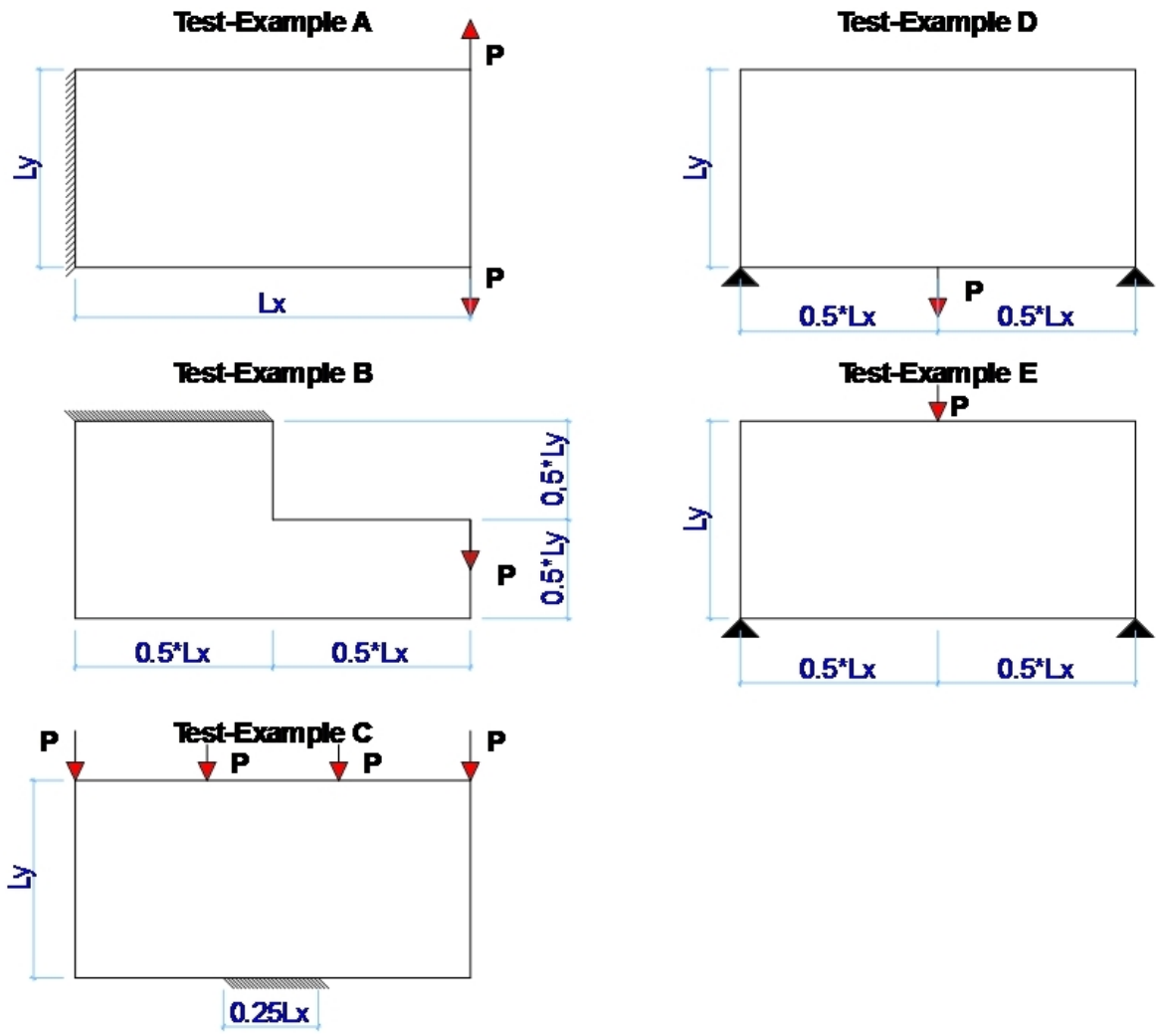


Fig. 5.23 Schematic representation of Test-Examples

5.25(a),(b),(c) and Fig. 5.25(d) respectively. In Test-Example C, CN-TOP achieved a reduction of computational time equal to 71.71% with respect to SIMP while the objective function value was increased by 0.19%. The final topology produced by SIMP alone, the one proposed by CN-TOP, the CNN proposed topology and the one exported from convolution can be seen in Fig 5.26(a),(b),(c) and Fig. 5.26(d) respectively. In Test-Example D, CN-TOP achieved a reduction of computational time equal to 72.40% with respect to SIMP while the objective function value was decreased by 0.05%. The final topology produced by SIMP alone, the one proposed by CN-TOP, the CNN proposed topology and the one exported from convolution can be seen in Fig 5.27(a),(b),(c) and Fig. 5.27(d) respectively. In Test-Example E, CN-TOP achieved a reduction of computational time equal to 67.87% with respect to SIMP while the objective function value was decreased by 0.25%. The final topology produced by SIMP alone, the one proposed by CN-TOP, the CNN proposed topology and the one exported from convolution can be seen in Fig 5.28(a),(b),(c) and Fig. 5.28(d) respectively. All data recorded for CN-TOP can be seen in Table 5.12.

Table 5.12 DLRM-TOP performance in Test-Examples A to E.

Test Example	SIMP			CN-TOP			Acceleration (%)	Obj. Function Value Reduction (%)
	Iterations	Obj. Function Value	Time	Iterations	Obj. Function Value	Time		
A	375	242.96	1336.94	108	242.36	429.10	67.90	0.25
B	507	156.77	2025.16	101	155.76	501.74	75.22	0.64
C	517	111.98	2243.77	142	112.19	634.77	71.71	-0.19
D	336	21.67	1347.37	71	21.66	371.94	72.40	0.05
E	310	24.12	1050.74	86	23.85	337.56	67.87	1.11

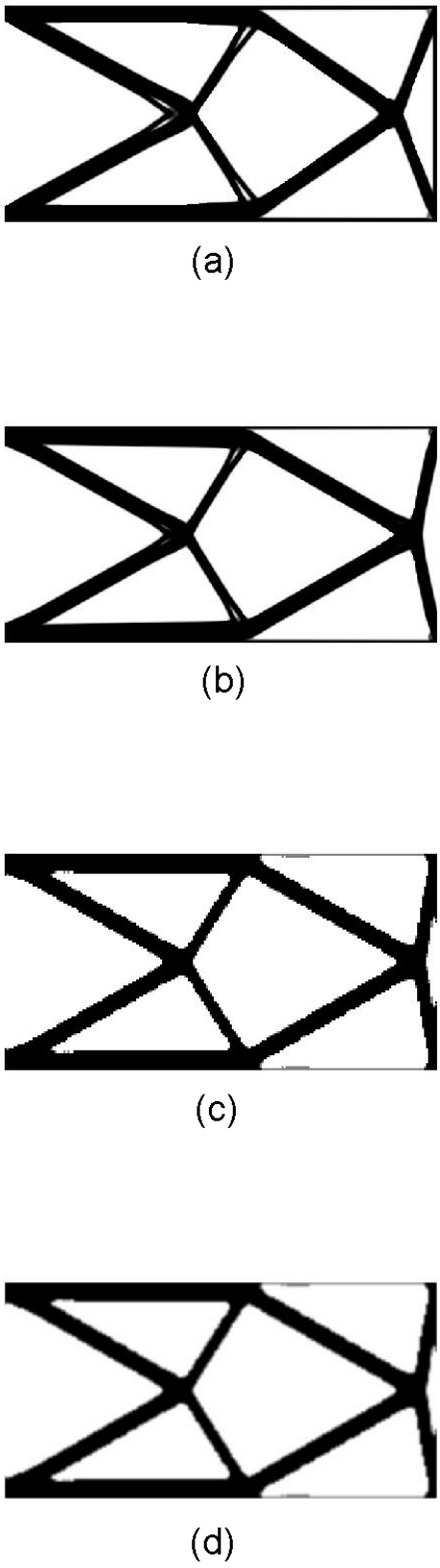


Fig. 5.24 Optimized domain for Test-Example A for each discretization: (a) SIMP output, (b) CN-TOP output, (c) CNN output, (d) Convolution output.

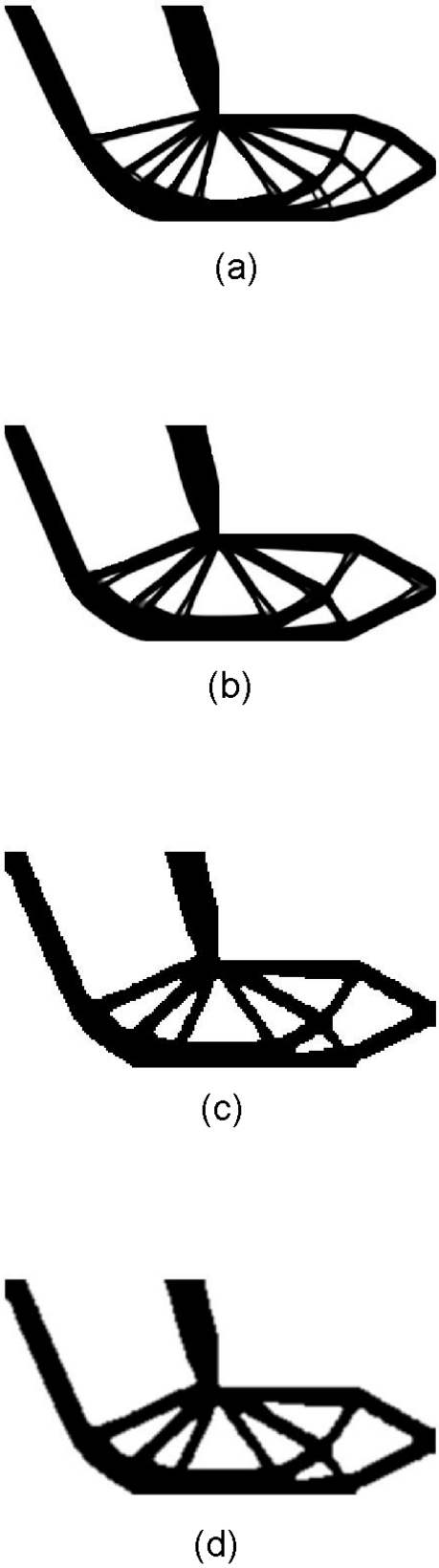


Fig. 5.25 Optimized domain for Test-Example B for each discretization: (a) SIMP output, (b) CN-TOP output, (c) CNN output, (d) Convolution output.

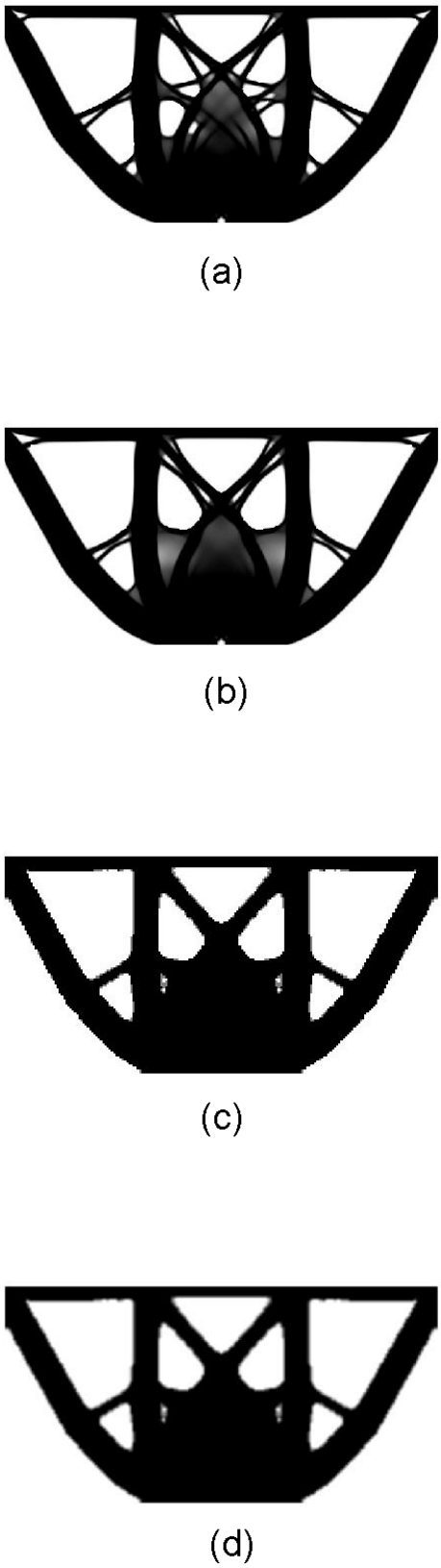


Fig. 5.26 Optimized domain for Test-Example C for each discretization: (a) SIMP output, (b) CN-TOP output, (c) CNN output, (d) Convolution output.

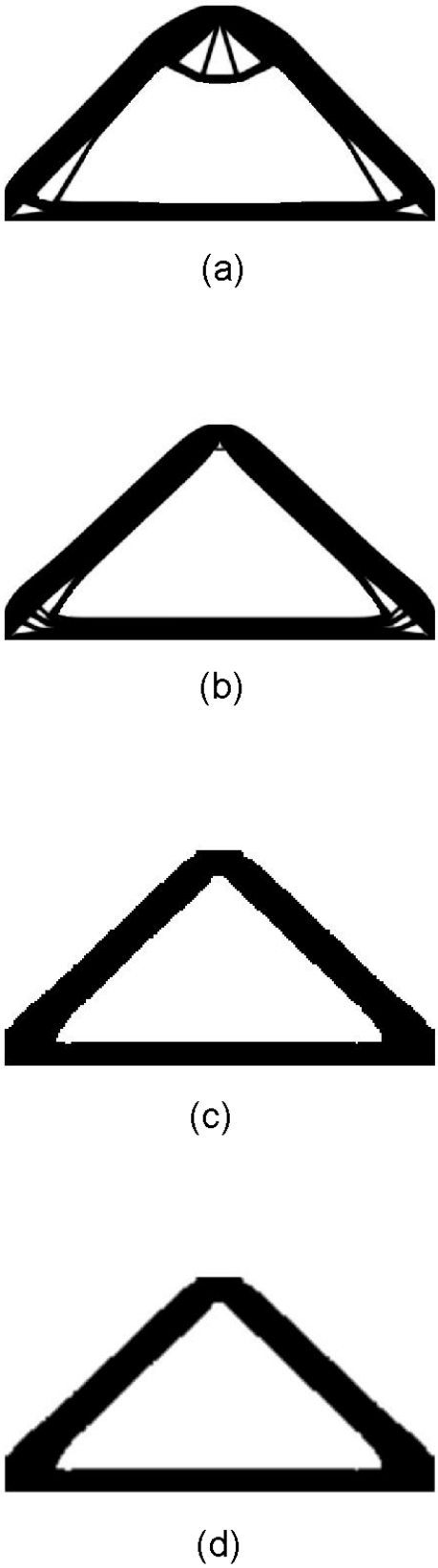
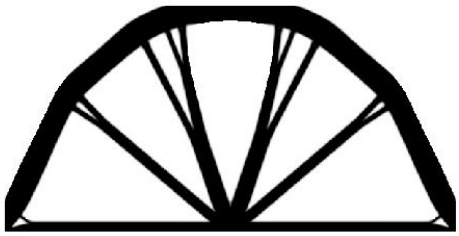
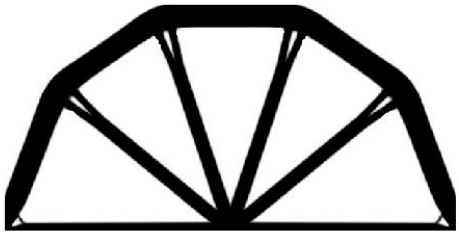


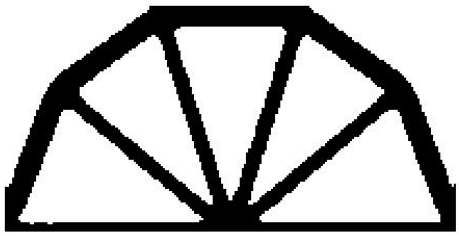
Fig. 5.27 Optimized domain for Test-Example D for each discretization: (a) SIMP output, (b) CN-TOP output, (c) CNN output, (d) Convolution output.



(a)



(b)



(c)

(d)

Fig. 5.28 Optimized domain for Test-Example E for each discretization: (a) SIMP output, (b) CN-TOP output, (c) CNN output, (d) Convolution output.

Chapter 6

Generative design based on Deep Learning and Optimization

6.1 Generative design

Generative design (GD) can be described as a methodology of creating a population of designs with respect to architect/engineer-defined constraints with the use of one or more algorithms in an iterative manner. GD works as a designer support tool which helps in expanding possible prototypes for the small number of designer's experience generated ones to the large number of computer generated ones. It differs from shape-generating optimization methods like topology optimization as GD focuses on producing a population of feasible solutions with respect to constraints and criteria while optimization methods focus on discovering a single optimal solution according to architect/engineer-defined constraints. Usually, GD is performed with the use of nature-mimicking procedures like growth, evolution, etc. In a short historical review, it can be witnessed that GD methods were introduced around 1975 with an effort to algorithmically mimic design patterns in nature like leafs [130] and dendritic shapes [140]. In the following years up to today, several other approaches have been proposed [5, 20, 69, 80, 157]. The main sectors where GD is applicable are:

- Product manufacturing
- Automotive industry
- Aerospace industry
- Architectural and construction industry

Some widely used methods in GD in the past in architecture are [195]:

- Cellular automata [214, 219]
- L-systems [130]
- Shape grammar [199]

In the recent years commercial software from well-known companies in the field of engineering software have been presented with some of them being:

- Autodesk's Fusion 360©
- Altair's Inspire©

It is also worth pointing out that commercial applications of GD are now a reality. For example, a part of AIRBUS 320© has been developed with a GD technique based on two algorithms, Slime Mold and Bone Growth algorithm [7], also the office floor planning of Autodesk's Mars© building in Toronto was also designed via GD [8, 212]. Regarding manufacturing industry, the combination of GD and 3D printing is already being used in several product types ranging from furniture to bicycle parts and many more.

In the current chapter of the dissertation, a new method for Generative Design inspired by the combination of SIMP approach and methodologies developed in the framework of this thesis (i.e. DL-SCALE and DLRM-TOP) is presented. This method presents a significant ability to propose different designs in a automated way through topology optimization and deep learning while the only necessary architect/engineer effort is the definition of the design domain size, loading and support conditions and preferred volume fraction (which are also parameters that will affect the aesthetic of the designs obtained and represent the architect/engineer intervention).

6.2 DzAI$\mathbb{N}$ - Generative design by Deep Learning

In the current section a method for computer generated design, Dz**AI**$\mathbb{N}$, is described. This method is based on the form-finding principles of SIMP and the differentiating shape generation strength of DL-SCALE and DLRM-TOP. SIMP has the ability to define the optimal volume distribution inside a domain with respect to structural performance, worth mentioning that the proposed methodology is not limite to SIMP only it is straight forward to implemented either with Level-Set or BESO approaches. As a gradient-based method, a domain with specific loading and support conditions and identical SIMP parameters (filter, volume fraction, etc.) will always converge to the same shape. The goal of generative design is to offer the designer the ability to quickly produce a large number of initial designs without

any architect/engineer interference apart from providing the problem definition parameters. DL-SCALE and DLRM-TOP have the ability to generate alternate shapes according to the data used as input. DzAI$\mathbb{N}$, presented in detail below, exploits the combination of the above features of SIMP, DLRM-TOP and DL-SCALE.

6.2.1 DzAI$\mathbb{N}$ method description

An STO problem, as previously mentioned, is characterized by the following model properties of TOP^f:

- Domain dimensions (L^f_{xyz})
- FE population (ne^f)
- Support conditions (S^f_c)
- Loading conditions (P^f_c)
- Target volume (V^f_t)

From the generative design aspect, these properties represent the user input from which several shapes must be produced. As described in the DL-SCALE and DLRM-TOP description sections, the SIMP-generated premature and final topologies of five reduced models with different ne^fs can assist in finding the optimal topology of a model with significantly denser mesh. It can be witnessed in the results of DLRM-TOP and DL-SCALE that although the objective function value of DL-based generated topologies is similar to the one generated by SIMP, worth mentioning that the shapes of the two DL methods present differences with the ones produced by SIMP. This observation led to the creation of DzAI$\mathbb{N}$ method.
According to DzAI$\mathbb{N}$, when designing a specific shape, the model properties of TOP^f need to be defined by the architect/engineer. Accordingly, the model properties of $TOP^R_1, TOP^R_2, TOP^R_3$, TOP^R_4 and TOP^R_5 are defined. SIMP is applied to each of the TOP^R_i and 35 iterations are performed. Record is kept of the density matrix D at 5 different numbers of performed iterations resulting to D^i_j where $j \in [5, 10, 15, 20, 25, 35]$ and $i \in [1, 2, 3, 4, 5]$. Those D^i_j matrices are extrapolated to the mesh dimensions of TOP^f as described in DL-SCALE and DLRM-TOP methods, creating $D^{i(F)}_j$.
Accordingly, six different inputs T^k_z are created where $k \in [1, 2, 3, 4, 5, 6]$ by using the

extrapolated density matrices for each i. T_z^k can be seen in Eq. 6.1.

$$T_z^k = \begin{bmatrix} T_{1,1}^{(d)} & T_{1,2}^{(d)} & \cdots & T_{1,ne_x^{(f)}}^{(d)} \\ T_{2,1}^{(d)} & T_{2,2}^{(d)} & \cdots & T_{2,ne_x^{(f)}}^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ T_{ne_y^{(f)},1}^{(d)} & T_{ne_y^{(f)},2}^{(d)} & \cdots & T_{ne_y^{(f)},ne_x^{(f)}}^{(d)} \end{bmatrix} \tag{6.1}$$

$$where \quad T_{i,j}^{(d)} = \left[\{d_{i,j}\}_{D_j^{i(F)}}^k, \{d_{i,j}\}_{D_j^{i(F)}}^k, \ldots, \{d_{i,j}\}_{D_j^{i(F)}}^k \right], \; and$$

$$i \in [1, ne_y^f], \; j \in [1, ne_x^f]$$

Each of the six T_z^k is used as input for a trained DBN network as described in DLRM-TOP section. The output of the network is a proposed topology with ne^f FEs. The procedure continues by applying for different convolution filters in each T_z^k and four filtered $T_z^{k(Filt)}$ density matrices are acquired per k. As a result, a total of 24 proposed density matrices are acquired. Each one is then passed into SIMP for fine-tuning. The fine-tuning step includes 20 iterations. The final output of fine-tuning are 24 proposed shapes with respect to initial problem definition. It is worth pointing out that no changes where made in the sensitivity filter and the volume fraction in order to prove the capabilities of Dz**AI**$\mathbb{N}$ with no changes in the STO problem definition. It is obvious that by including changes in the above two parameters, the population of proposed shapes can increase rapidly. A flowchart of Dz**AI**$\mathbb{N}$ can be seen in Fig. 6.1.

6.2.2 DzAI$\mathbb{N}$ method test examples

Dz**AI**$\mathbb{N}$ performance is evaluated on four 2D test-examples known to literature. The FE populations used in reduced domains are equal to $[7,000, 10,000, 15,000, 20,000, 25,000]$ while the number of FEs in the final domain is equal to $75,000$.

In Test-Example A, the support conditions refer to two fixed joints placed at the two right corners of the domain and the loading conditions refer to two concentrated forces P along the x axis and applied in the left and right middle of the span in the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 60%.

In Test-Example B, the support conditions refer to fully fixed boundary condition along the x axis ending at the half of the x dimension and the loading condition refers to one

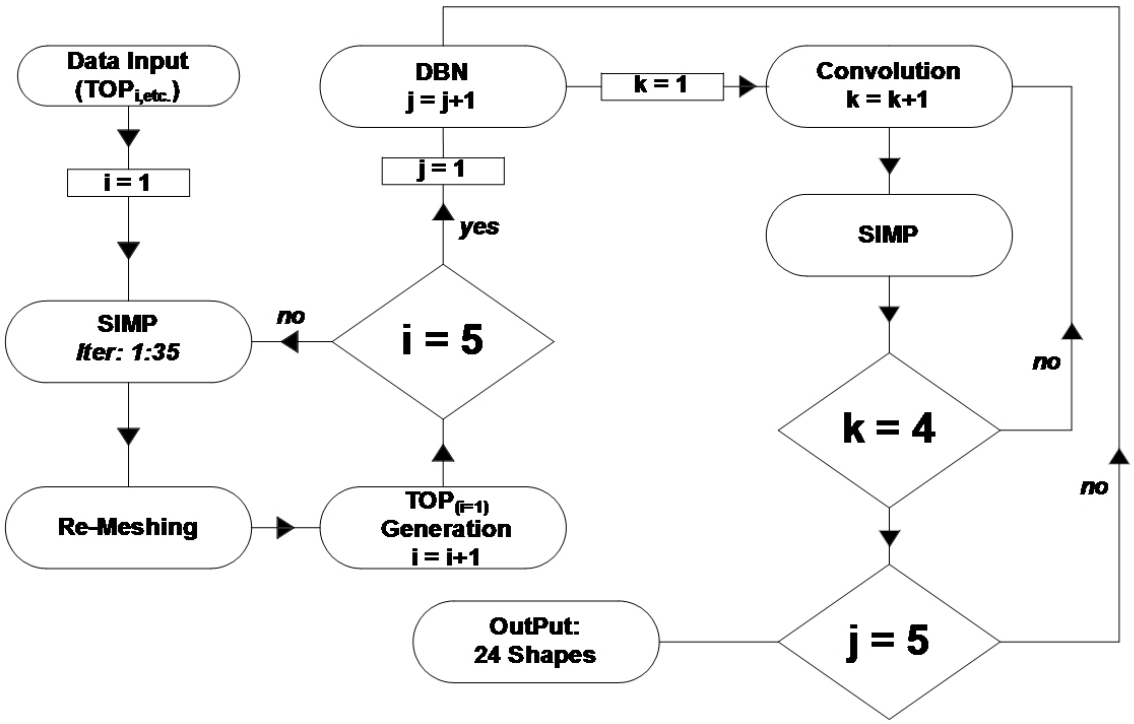


Fig. 6.1 Flowchart of DzAIℕ method.

concentrated force P along the y axis and applied in the fourth of the y dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 35%.

In Test-Example C, the support conditions refer to two fixed joints placed at both left and right lower end corners of the domain and the loading conditions refer to one concentrated force P along the y, applied at the half of the y dimension at the top of the structure. The ratio of ne_x to ne_y is equal to $1/3$ while the volume fraction is equal to 50%.

In Test-Example D, the support conditions refer to two fixed joints, the first is placed at the lower right corner of the domain the second at the 3/5 of the lower edge. The loading conditions refer to a distributed load P along the y axis and applied along the x dimension. The ratio of ne_x to ne_y is equal to 0.5 while the volume fraction is equal to 45%.

It is worth pointing out that a sensitivity filter with radius equal to 3 was chosen in all cases as SIMP filter while no symmetry is imposed. A schematic representation of the test-examples can be seen in Fig. 6.2.

6.2.3 DzAIℕ method results

In Test-Example A, 24 different designs generated by DzAIℕ are presented in Fig. 6.3a, 6.4b, 6.5c. The designs generated for Test-Examples B,C and D can be seen in Fig. 6.6a, 6.7b, 6.8c,

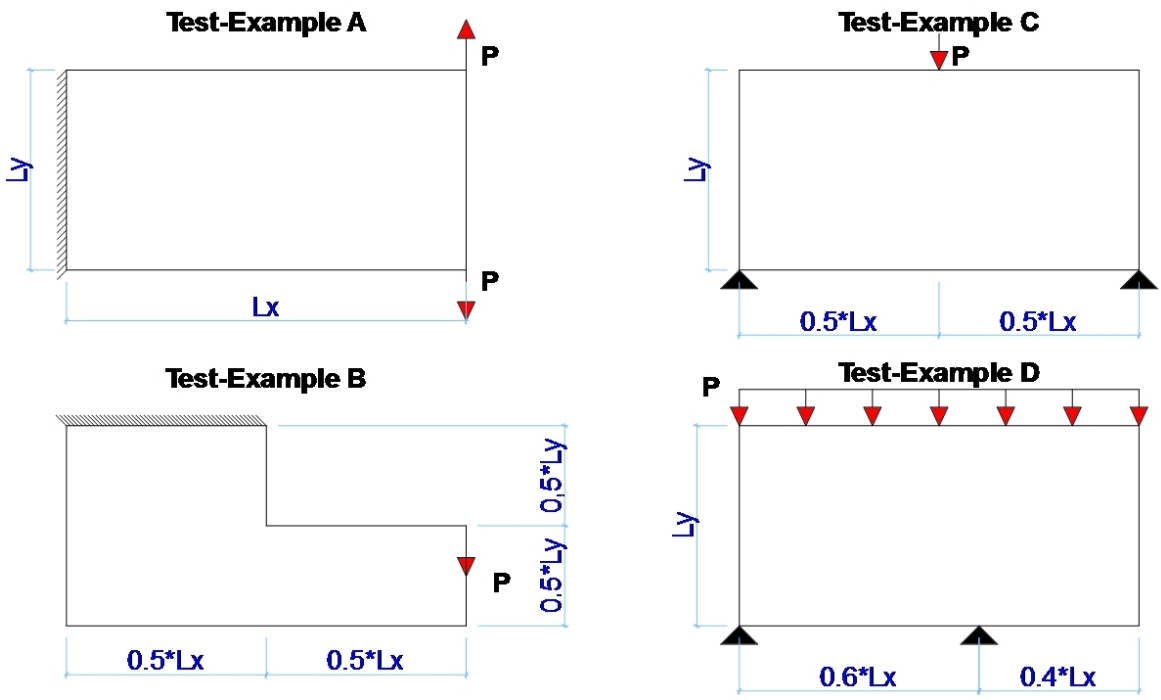


Fig. 6.2 Schematic representation of Test-Examples

Fig. 6.9a, 6.10b, 6.11c, and Fig. 6.12a, 6.13b, 6.14c, respectively. These figures present the variation of the equivalent solution generated by means of Dz**AI**$\mathbb{N}$.

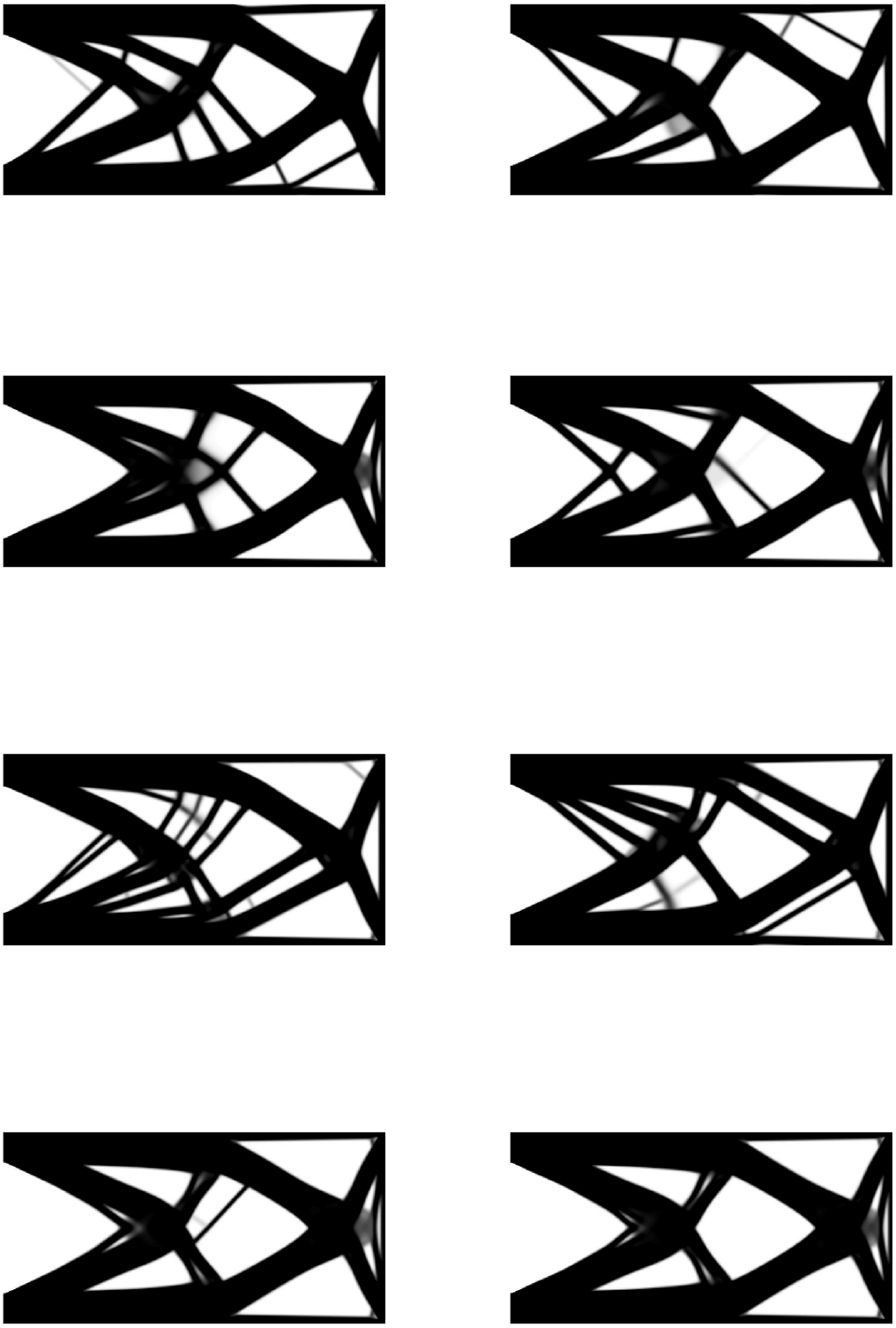


Fig. 6.3 a. Generated designs 1-8 for Test-Example A.

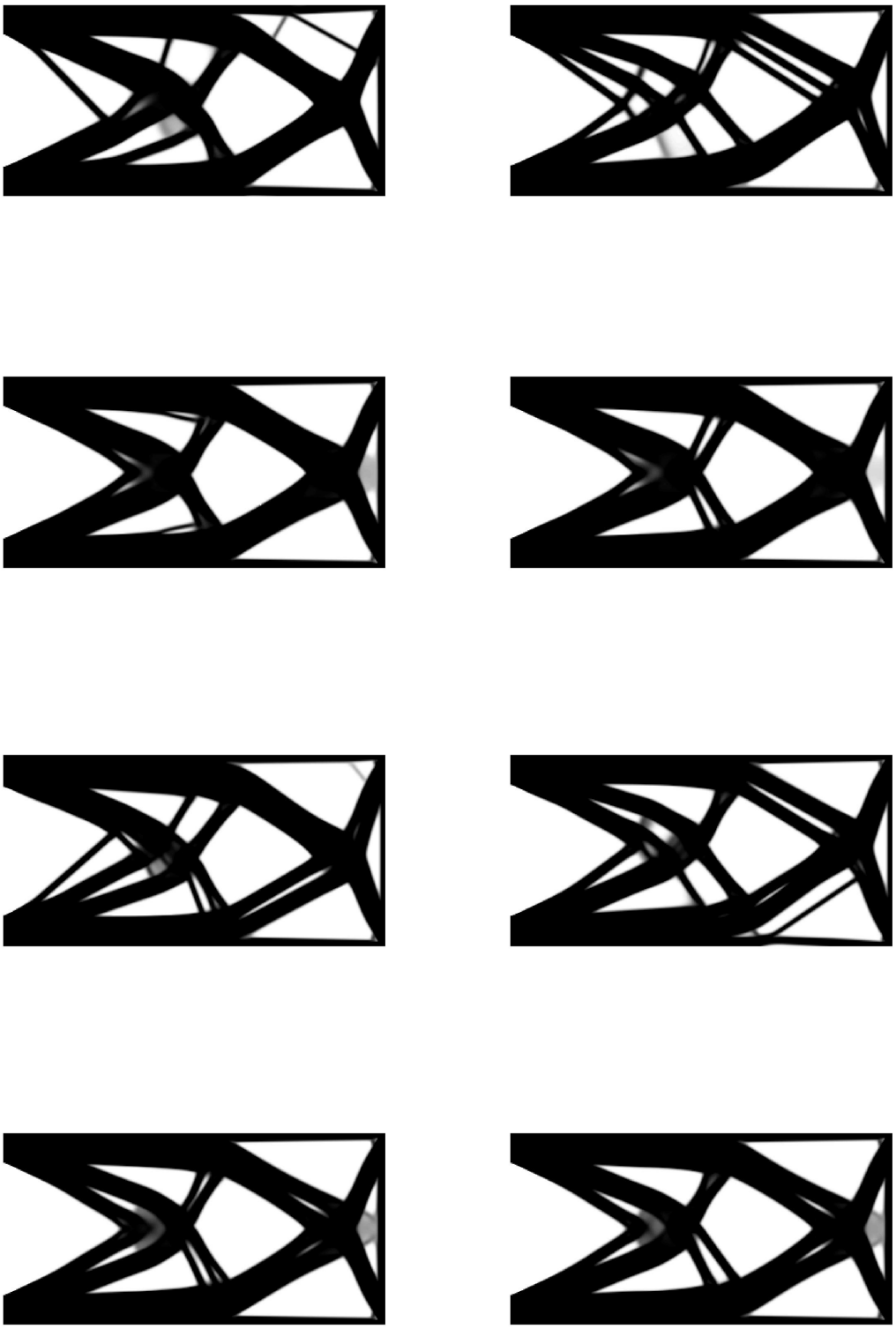


Fig. 6.4 b. Generated designs 9-16 for Test-Example A.

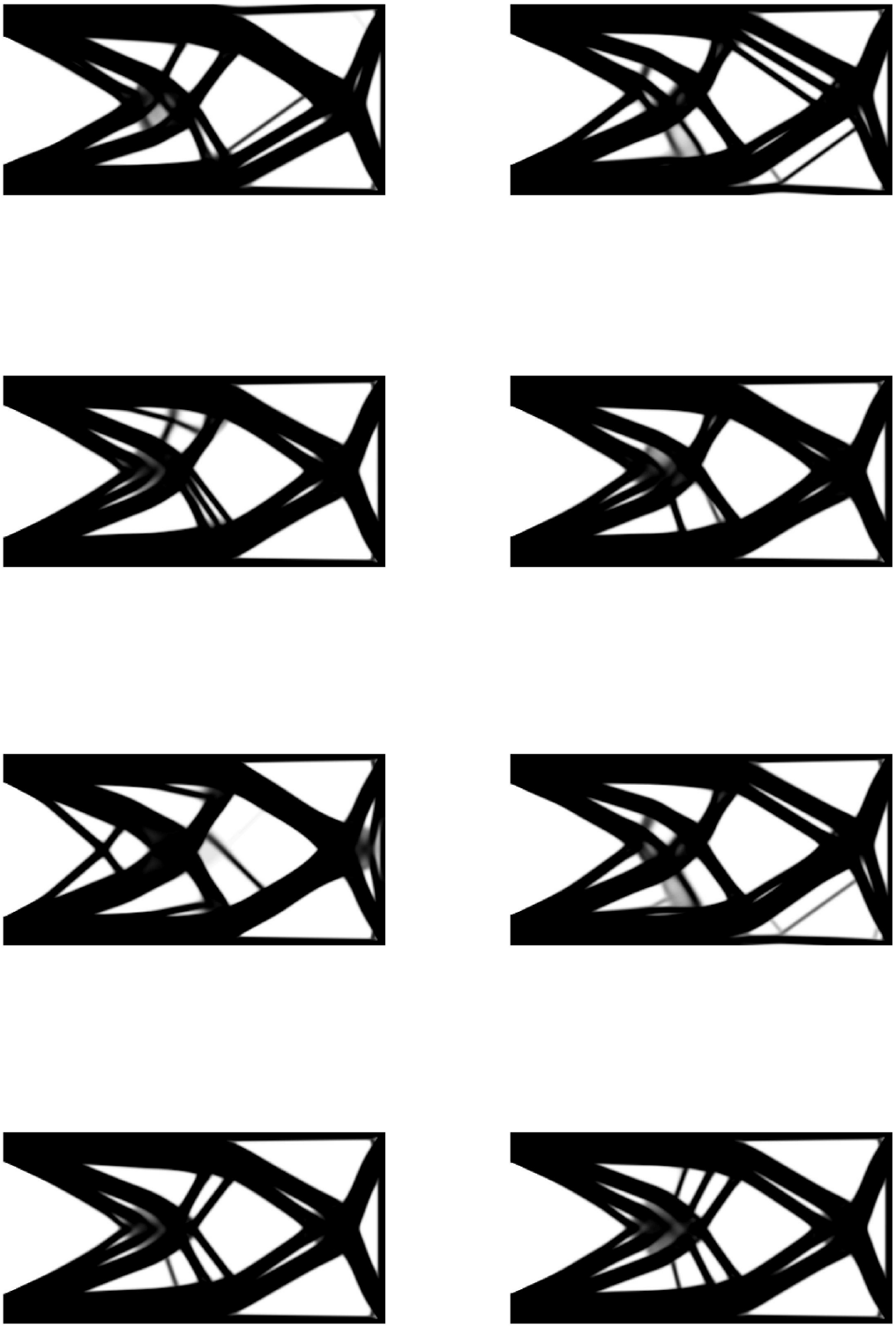


Fig. 6.5 c. Generated designs 17-24 for Test-Example A.

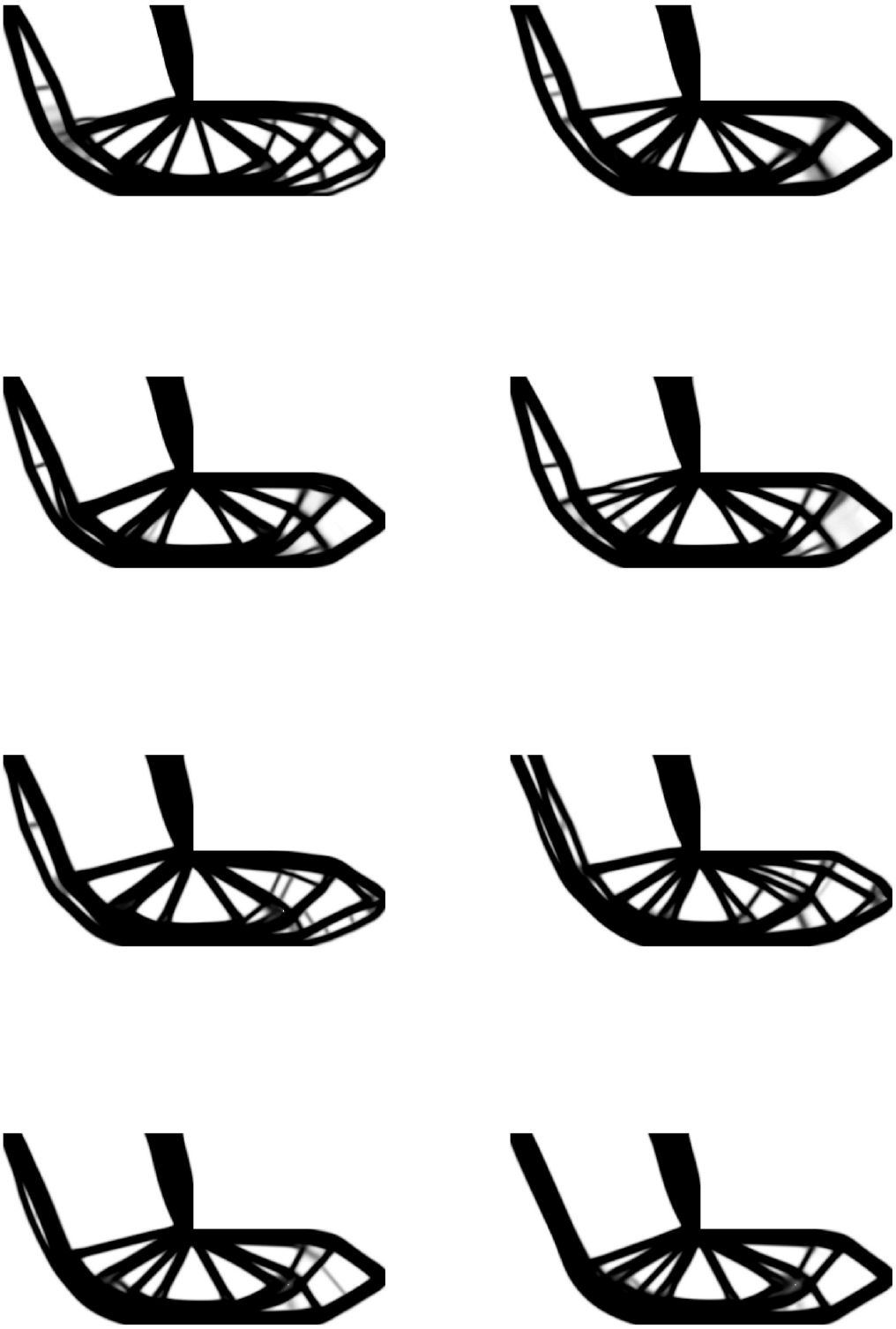


Fig. 6.6 a. Generated designs 1-8 for Test-Example B.

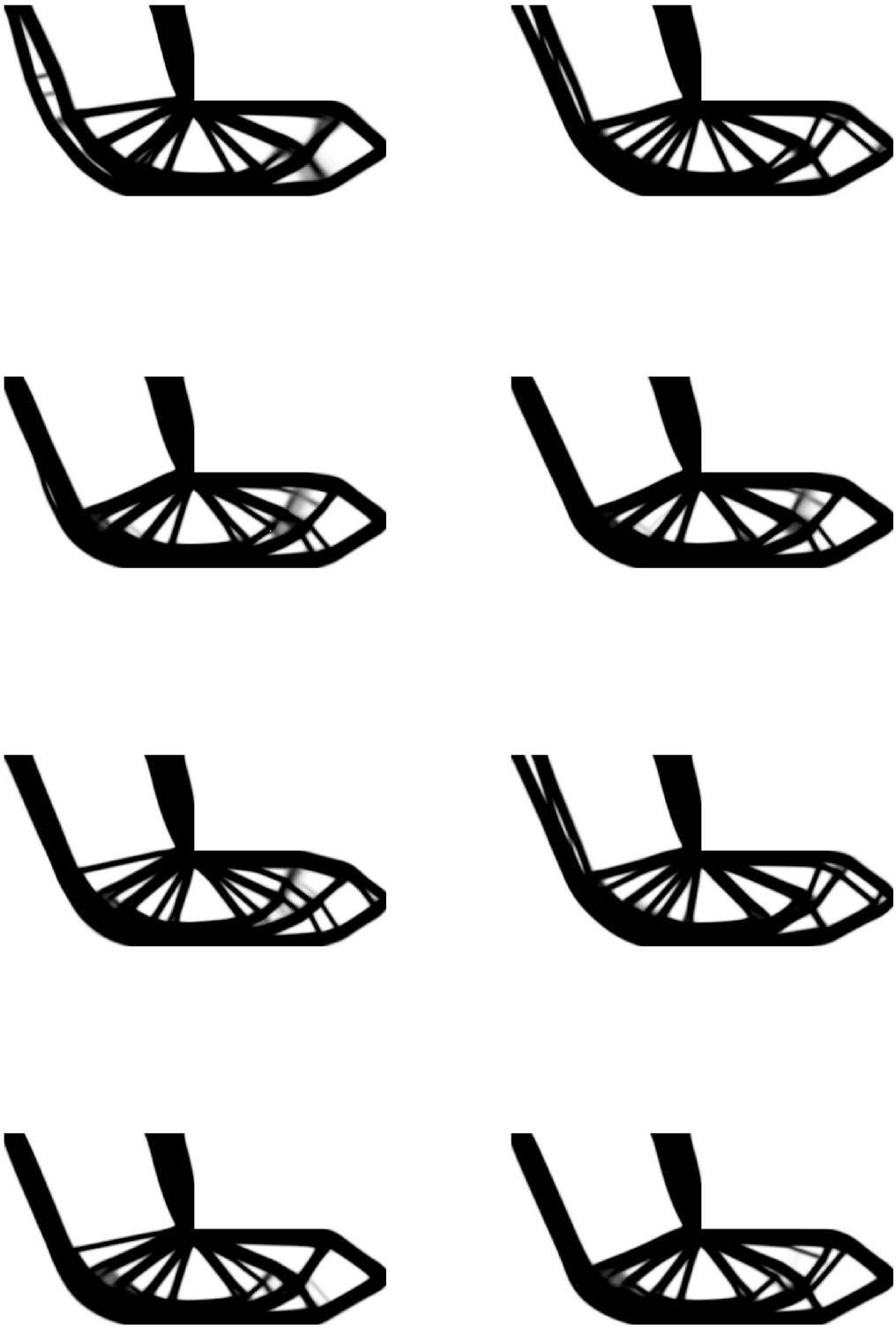


Fig. 6.7 b. Generated designs 9-16 for Test-Example B.

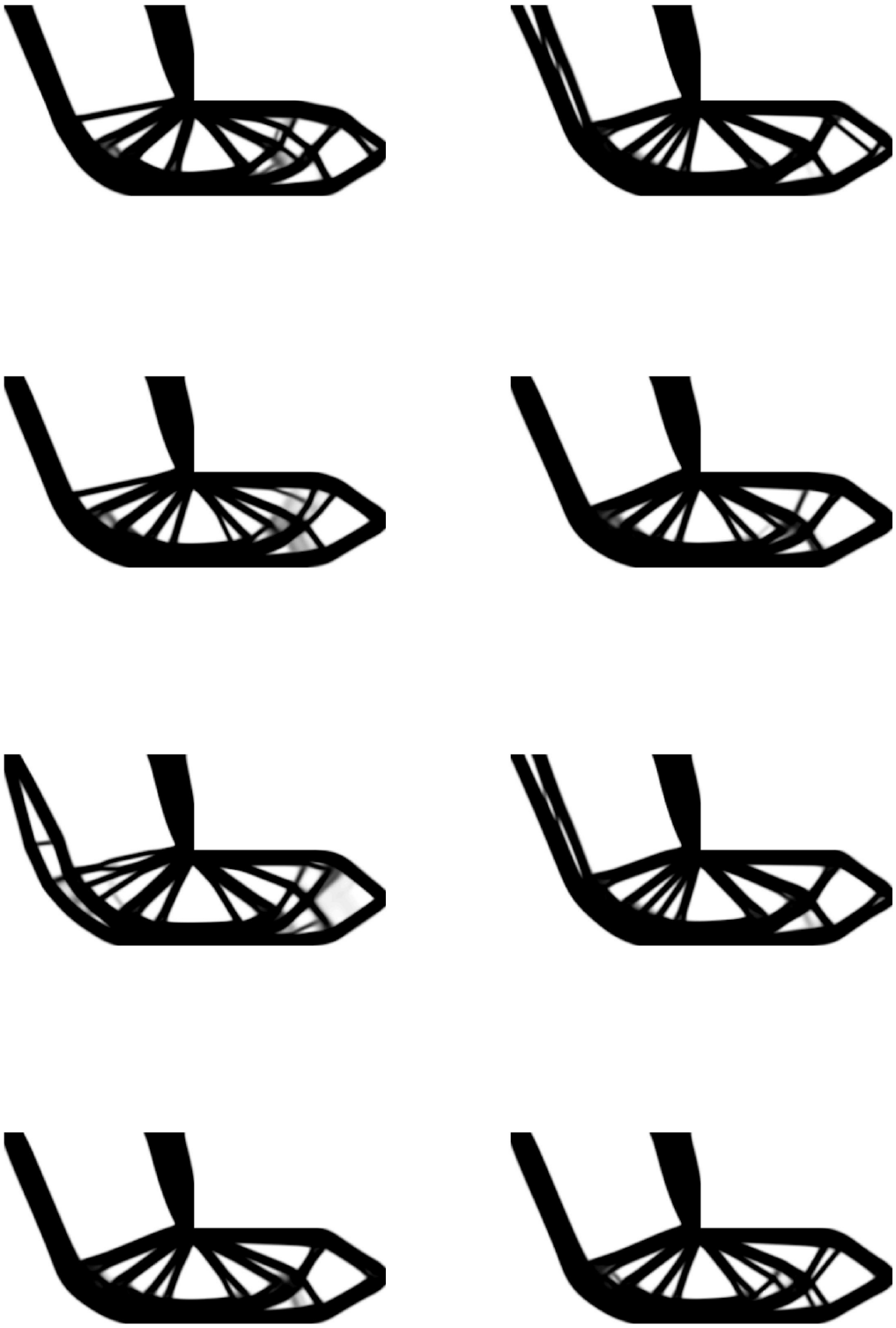


Fig. 6.8 c. Generated designs 17-24 for Test-Example B.

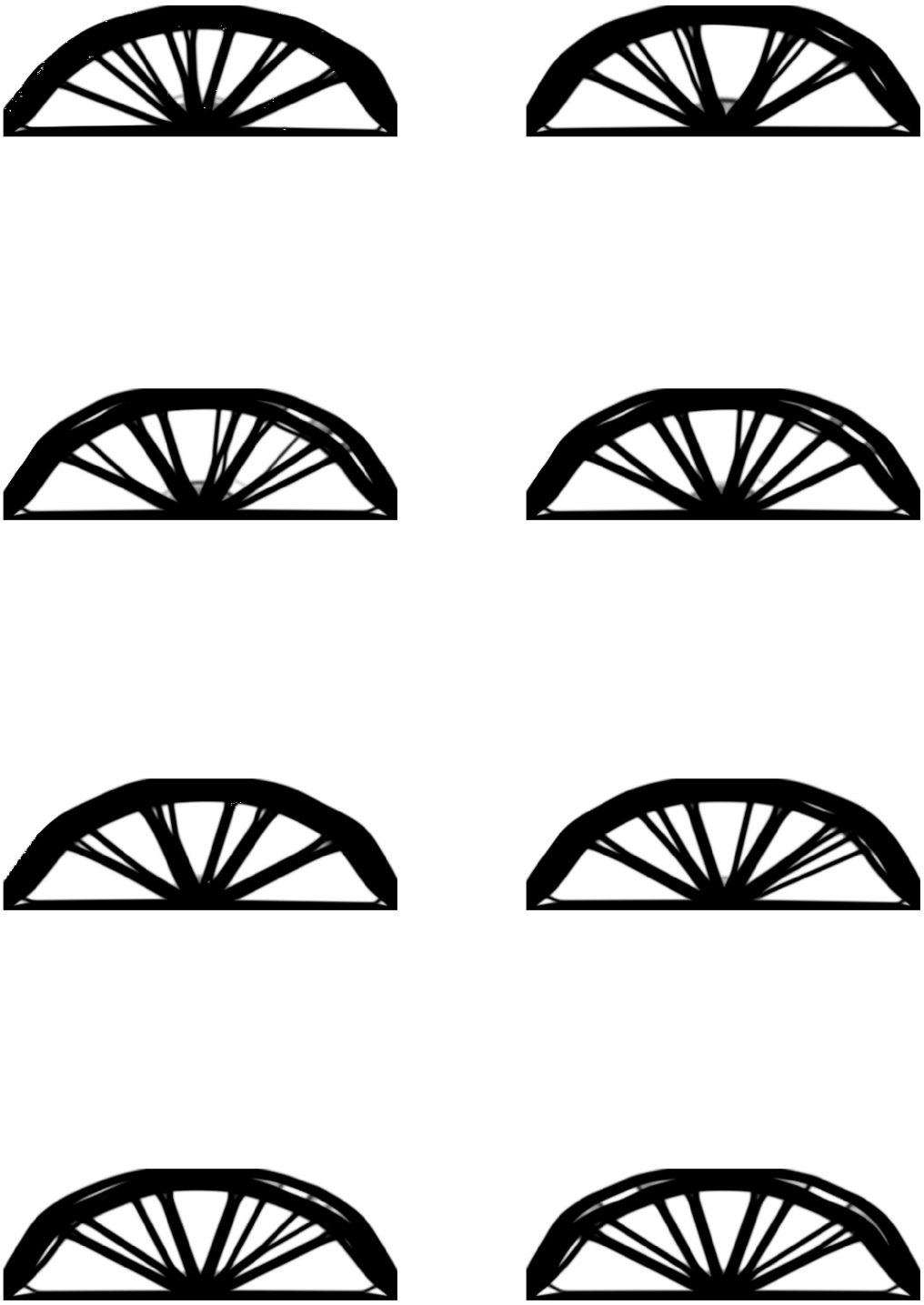


Fig. 6.9 a. Generated designs 1-8 for Test-Example C.

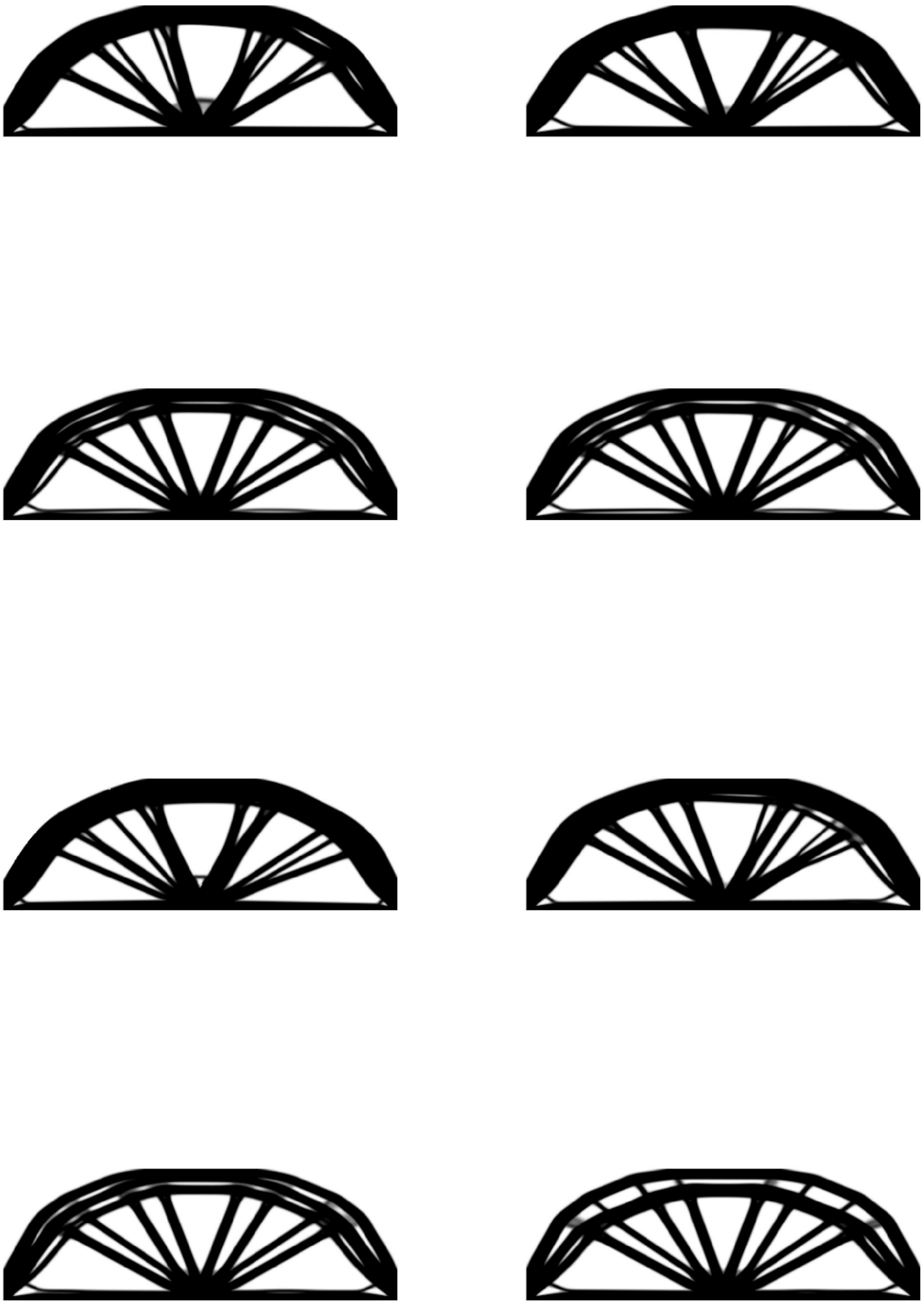


Fig. 6.10 b. Generated designs 9-16 for Test-Example C.

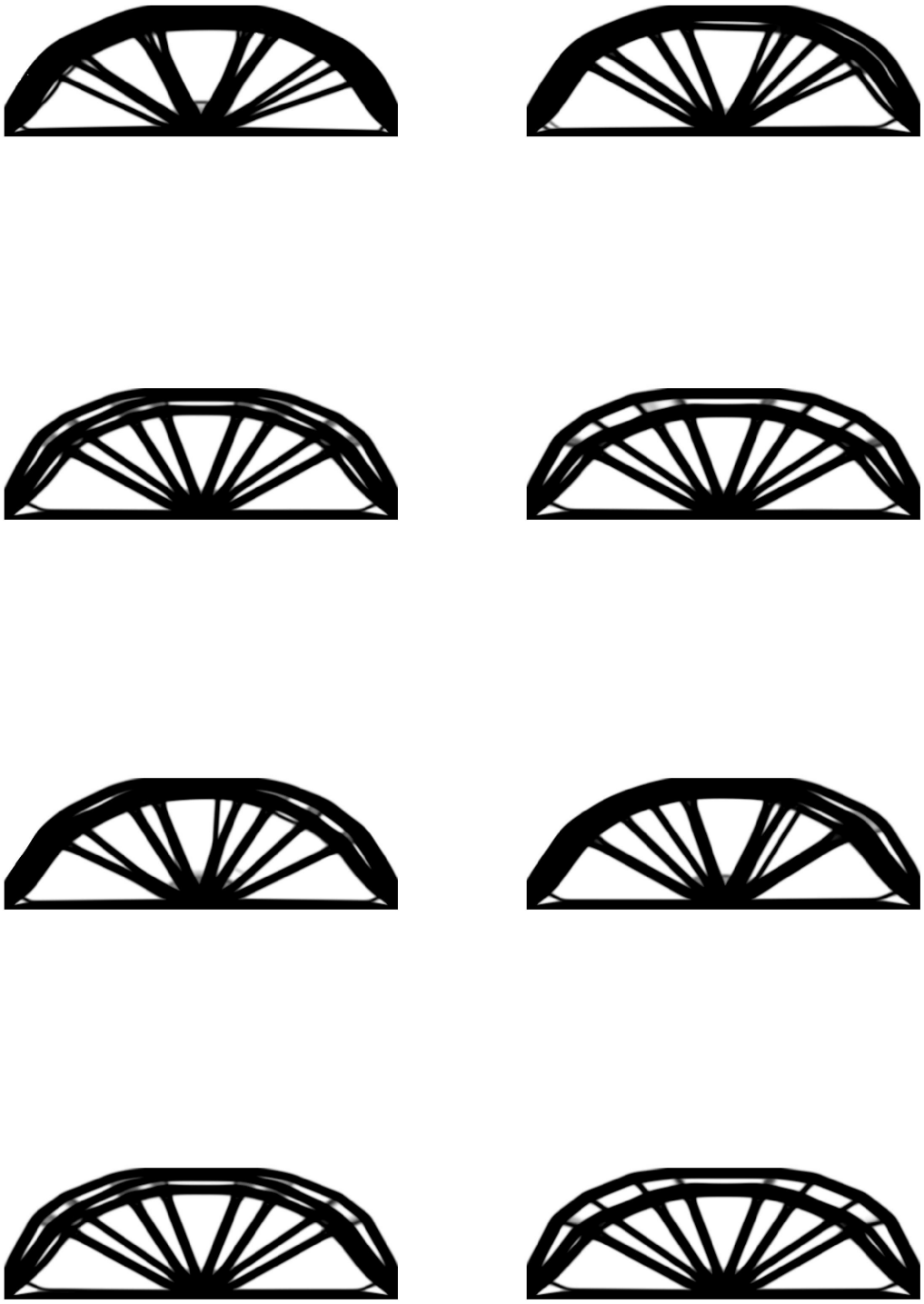


Fig. 6.11 c. Generated designs 17-24 for Test-Example C.

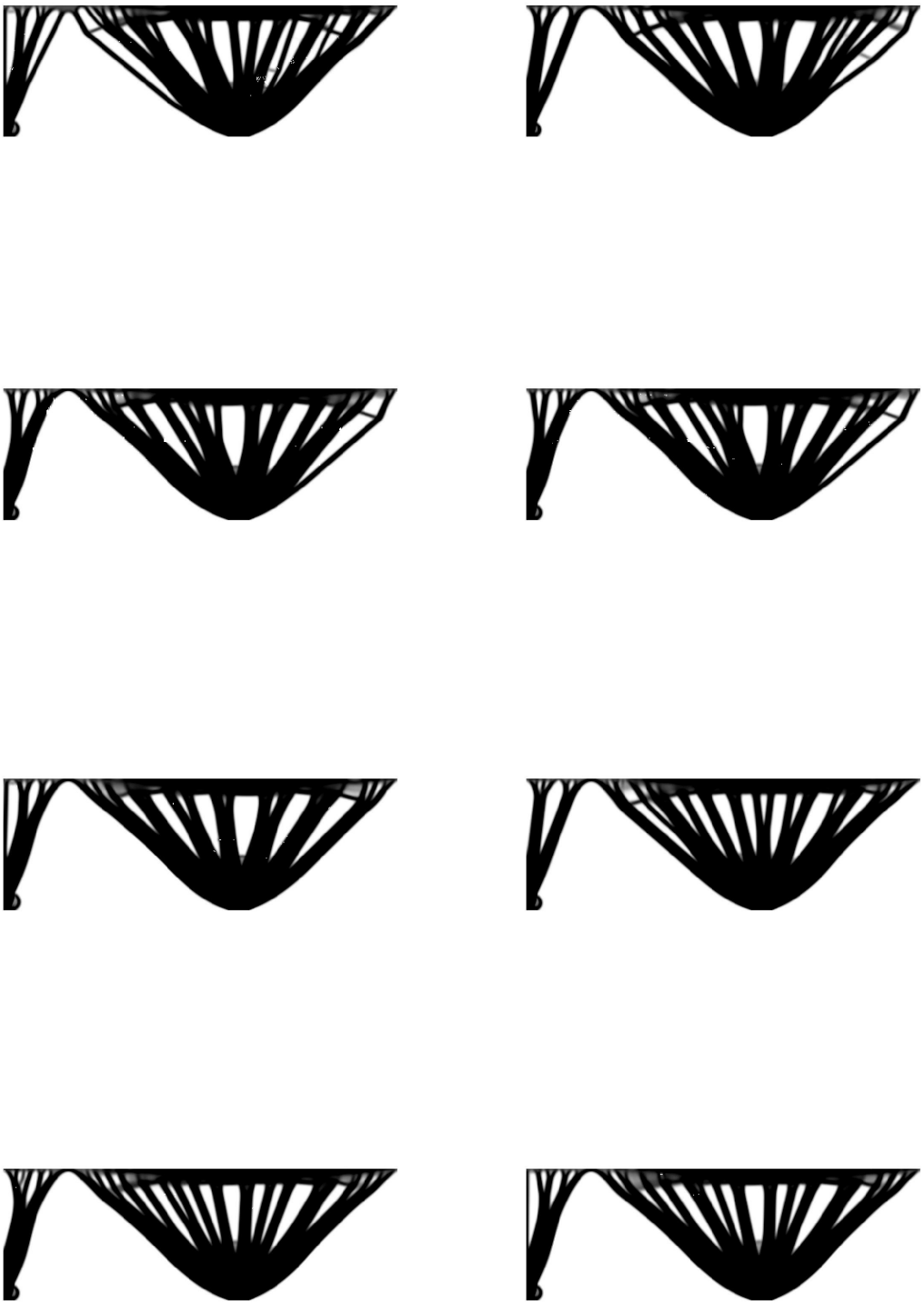


Fig. 6.12 a. Generated designs 1-8 for Test-Example D.

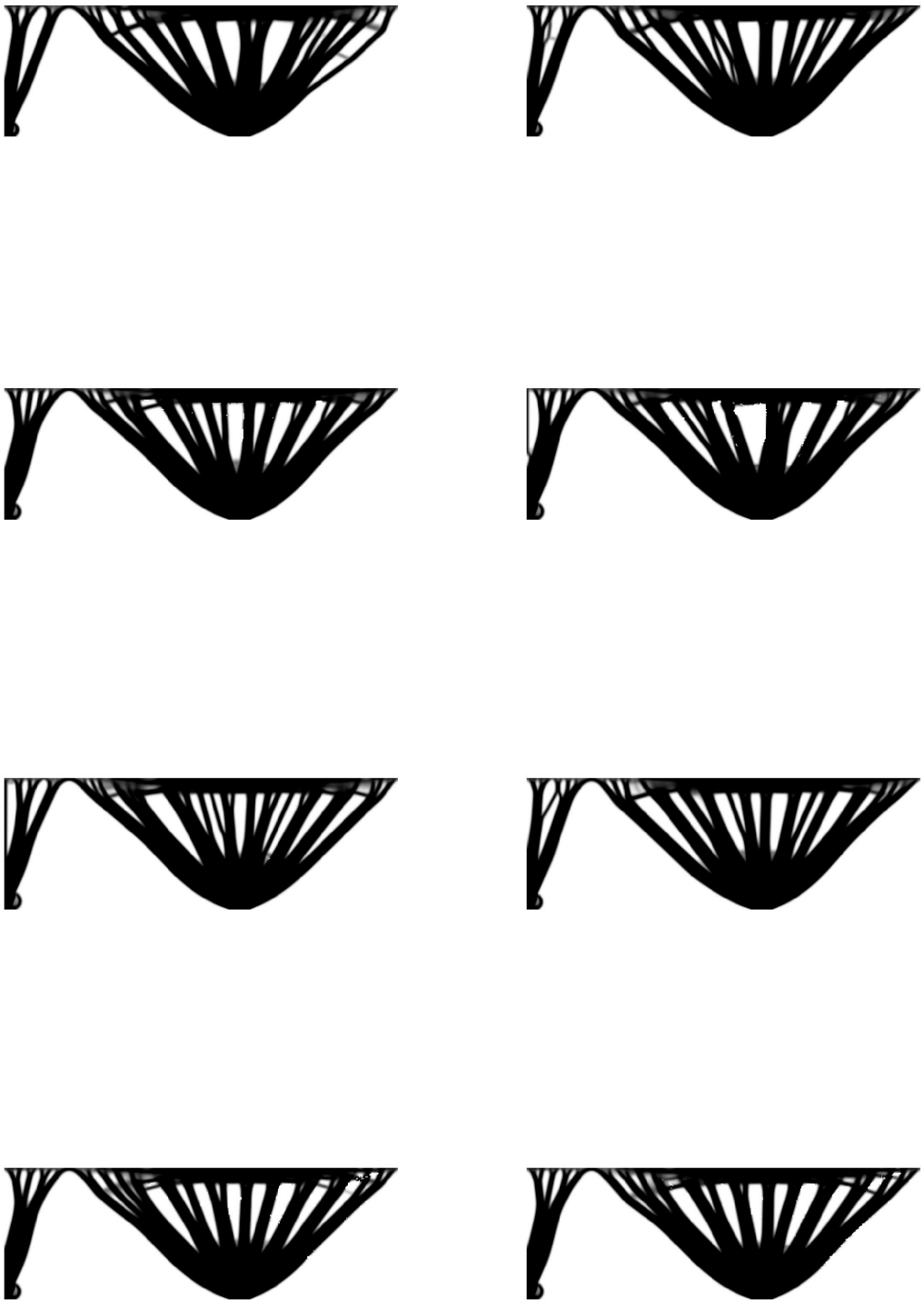


Fig. 6.13 b. Generated designs 9-16 for Test-Example D.

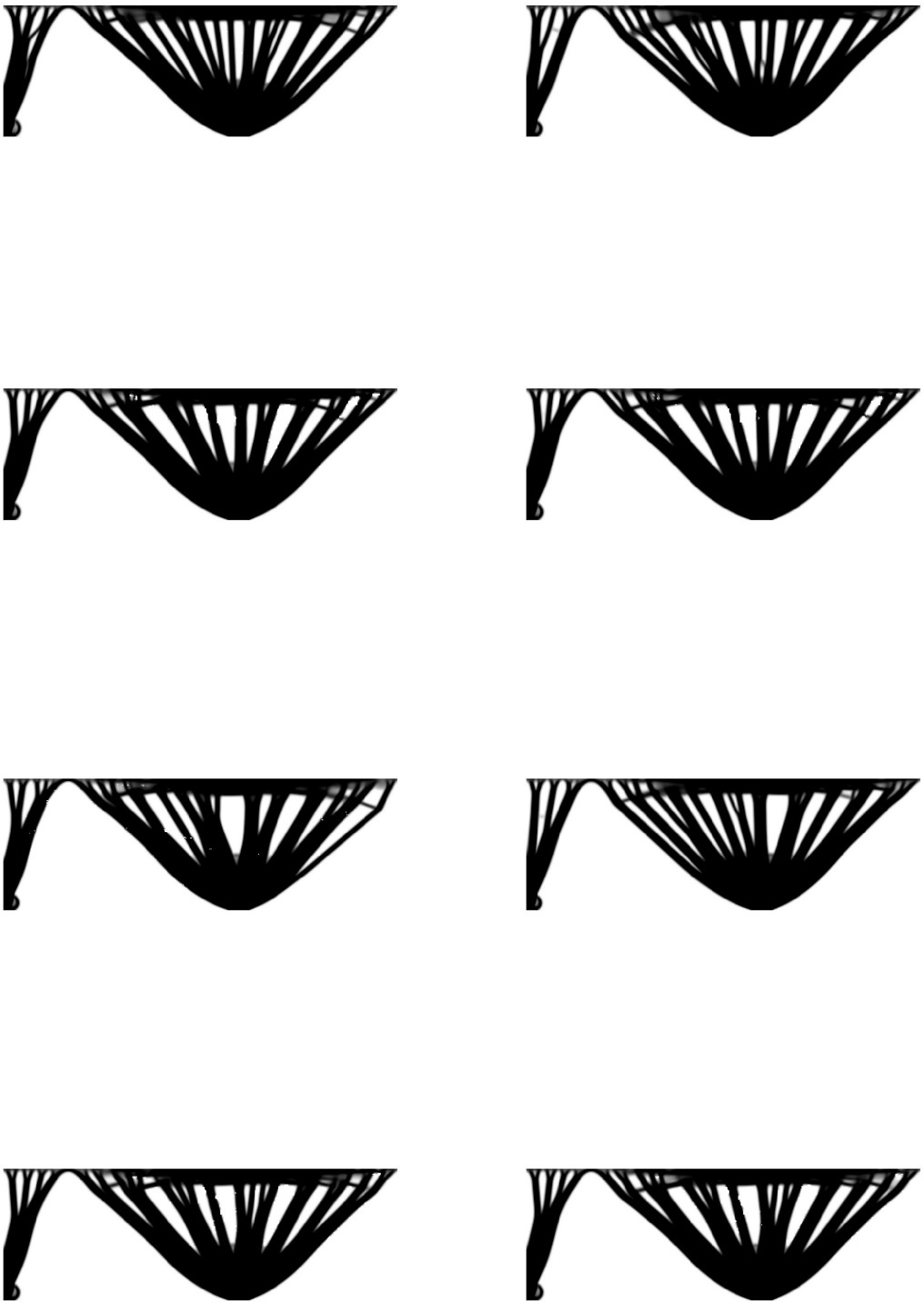


Fig. 6.14 c. Generated designs 17-24 for Test-Example D.

Chapter 7

Future Work

In the course of this dissertation, knowledge and experience was gained in the fields of metaheuristics, machine learning, topology optimization, generative design and reduced order modeling while generated work on these fields was presented previously. Dealing with these topics on an everyday basis, led to the presented work but it, more importantly, led to new ideas and areas for development and testing. According to the writer's opinion, deep learning approaches present important features that can be exploited in problems that structural engineers face in their everyday practice. These features need to be thoroughly examined in order to be able to find the specific approach that can be assisted by deep learning. Additionally, new methods of deep learning usage on civil engineering problems can be established.
As also discussed and presented previously, deep leaning has proven to be successful in applications of:

- Computational load reduction
- Shape diversification
- Massive data handling
- Discovery of higher order correlations in data

The above abilities were identified through applications presented in previous chapters of this dissertation. In the following part of the dissertation, a few thoughts on possible future work of the writer are denoted.

7.1 Deep Learning in Topology Optimization

Apart from the work presented previously in the application of deep learning techniques in TO problems, there are several aspects that could be further examined. These aspects involve:

- Improvement of training dataset used in the methods previously described.
- Creating an on-line application for DL-TOP.
- Applying DL-SCALE and DLRM-TOP in 3D topology optimization problems.
- Further examine possible applications of CNNs in topology optimization problems.

Improvement of training dataset

In DL-TOP, a DBN network is trained in discovering correlations between initial values of density of a FE and its final value. Two datasets were created and tested with respect to correct prediction and generality. As described previously, these two databases are different in terms of distribution of classes. In the training procedure of both databases, the success rate was in the range of 87% *to* 92%. According to the writer's opinion, further tweaking of DBN training parameters can be examined. By applying these two modifications, the success rate of training can be increased up to a maximum of 95% which will lead to even better improvement of DL-TOP performance. According to the writer's opinion, a rate higher to 95% cannot be achieved realistically as there will always be some elements that alter their density value in an unpredictable manner.

DL-TOP on-line

As previously presented, DL-TOP performs significantly well regardless of the type of topology optimization problem. Moreover, the executional time of DL-TOP is minimal, even on a serial mode of execution. In most cases, the significant part of DL-TOP from the perspective of time, is the execution of 36 iterations of the original topology optimization problems. The other two parts of the method (prediction of a close-to-final density per FE and SIMP fine-tuning) require much less time than the first one. By exploiting this advantage, an on-line application can be created for applying DL-TOP in any user-applied TO problem. It is also very important that DL-TOP does not need any information regarding geometry, loading conditions, support conditions, volume fraction, etc. of the user-defined TO problem. A user can provide the first 36 results of SIMP (density per iteration per FE) and an on-line pre-trained DBN will export the close-to-optimal density value for all FEs in the domain. This export can be fine-tuned by the user through SIMP application.

3D DL-SCALE and DLRM-TOP

In this dissertation, the two proposed model-upgrading methods (DL-SCALE and DLRM-TOP) are proven to be significantly successful in reducing computational load of 2D TO problems. The current formulation of these two methodologies is applicable in 2D TO problems. The basic concept of each of them is not restricted to 2D problems but it can also be applied to 3D ones where acceleration is expected to be even higher. In order for this to happen, a modification is needed in the way that the initial volume of the FEs of the final domain is chosen with respect to the ones of the less dense meshed domains. For example, geometric distance can again be the criterion but in this case, 3D coordinates must be used. Additionally, the convolution part of both methods also needs to be modified in order to be applicable on 3D domains. Both the above changes are necessary but doable at the same time. As a result, it is the writer's opinion that DL-SCALE and DLRM-TOP can be successfully applied to 3D TO problems as well.

Exploit CNN applications in Topology optimization

While CNNs are becoming very successful in several complex real-life applications, they are mostly used in image-related tasks, mainly due to their architecture. Although TO presents an image-related aspect, as a form-finding method, further applications of CNNs can be examined which are not image-based. New CNN architectures are being proposed with increased capabilities and input characteristics. As in structural engineering, almost no real-life tasks are two-dimensional, it is worth looking for new features in CNNs that can exploit their advantage of very deep architectures for assisting researchers in the field of structural engineering.

7.2 Conceptual Design and 3D printing

In Chapter 6, Dz**AI**$\mathbb{N}$, a DL/Optimization approach on Conceptual Design was presented. With this method, a large number of initial designs can be produced in a very short time while the designer can select the most preferable one(s) and edit them according to his insights and intuition. TO has been successfully applied in massive production of mechanical parts in several structures ranging from car to airplane parts. 3D printing has also assisted in automating the procedure of exporting the designer's model to a usable part. For various, mainly technical and economical reasons, this practice has not been yet applicable to civil engineering structures.

Recent advances in 3D printing are bringing metal structures and fiber-reinforced concrete

closer to structural engineering. As this major gap is being bridged, the exploitation of TO and conceptual design in civil engineering is approaching every-day practices. According to the writer's belief, this field of research will attract a lot of interest in the following years. TO, DL and 3D printing are not only necessary but almost obligatory in this research field. Either in terms of pre-constructed parts or free-form structures, the above methodologies will be proven very useful.
As a result, the writer is very interesting in working on defining a prototype framework under which the previously described methodologies will be used in a collaborative manner in the field of civil engineering constructions.

7.3 Deep Learning in Dynamic Analysis of Structures

According to the presented work regarding Deep Learning and reduced order models, there are significant capabilities in this approach. Dynamic structural analysis represent also a computationally demanding procedure. An attempt to accelerate the procedure through deep learning will be examined. The computational load of dynamic analysis depends on the number of nodes in the structural system examined and the number of seismic accelerations examined. According to the writer's opinion, a DL approach can be examined on two levels:

- Reduce dimension of acceleration time-series.
- Reduce number of nodes according to seismic behavior.

Both these approaches will be examined in the future.

References

[1] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169.

[2] Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine learning*, 6(1):37–66.

[3] Allaire, G., Jouve, F., and Toader, A.-M. (2002). A level-set method for shape optimization. *Comptes Rendus Mathematique*, 334(12):1125–1130.

[4] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. S., and Sigmund, O. (2011). Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16.

[5] Attar, R., Aish, R., Stam, J., Brinsmead, D., Tessier, A., Glueck, M., and Khan, A. (2009). Physics-based generative design.

[6] Audet, C. and Dennis, Jr., J. E. (2006). Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. on Optimization*, 17(1):188–217.

[7] Autodesk (2018a). Customer stories: Airbus. https://www.autodesk.com/customer-stories/airbus. Accessed: 2019-03-15.

[8] Autodesk (2018b). Space-planning. https://www.autodesk.com/autodesk-university/article/Generative-Design-Architectural-Space-Planning-2018. Accessed: 2019-03-15.

[9] Avtzis, D., Arthofer, W., Stauffer, C., Avtzis, N., and Wegensteiner, R. (2010). Pityogenes chalcographus (coleoptera, scolytinae) at the southernmost borderline of norway spruce in greece. *Entomologia Hellenica*, 19:3–13.

[10] Bell, M. G. H. (2000). A game theory approach to measuring the performance reliability of transport networks. *Transportation Research Part B: Methodological*, 34(6):533–545.

[11] Bellman, R. E. (1957). *Dynamic Programming*. Courier Dover Publications.

[12] Bendsøe, M. P. (1989). Optimal shape design as a material distribution problem. *Structural optimization*, 1(4):193–202.

[13] Bendsøe, M. P. and Kikuchi, N. (1988). Generating optimal topologies in structural design using a homogenization method. *Computer methods in applied mechanics and engineering*, 71(2):197–224.

[14] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

[15] Bertsekas, D. (1999). *Nonlinear Programming*. Athena Scientific.

[16] Borrvall, T. and Petersson, J. (2001). Large-scale topology optimization in 3d using parallel computing. *Computer methods in applied mechanics and engineering*, 190(46-47):6201–6229.

[17] Bourdin, B. and Chambolle, A. (2003). Design-dependent loads in topology optimization. *ESAIM: Control, Optimisation and Calculus of Variations*, 9:19–48.

[18] Brighton, H. and Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172.

[19] Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90.

[20] Caldas, L. (2006). Gene_arch: an evolution-based generative design system for sustainable architecture. In *Intelligent computing in engineering and architecture*, pages 109–118. Springer.

[21] Carreira-Perpinan, M. A. and Hinton, G. E. (2005). On contrastive divergence learning. In *Aistats*, volume 10, pages 33–40. Citeseer.

[22] Cauchy, A. (1847). Methodes generales pour la resolution des systemes d' equations simultanees. *C.R. Acad. Sci. Par.*, 25:536–538.

[23] Cha, Y.-J., Choi, W., and Büyüköztürk, O. (2017). Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 32(5):361–378.

[24] Challis, V. J., Roberts, A. P., and Grotowski, J. F. (2014). High resolution topology optimization using graphics processing units (gpus). *Structural and Multidisciplinary Optimization*, 49(2):315–325.

[25] Chandrinos, S. K. and Lagaros, N. D. (2018). Construction of currency portfolios by means of an optimized investment strategy. *Operations Research Perspectives*, 5:32 – 44.

[26] Chang, S. E. and Nojima, N. (2001). Measuring post-disaster transportation system performance: the 1995 kobe earthquake in comparative perspective. *Transportation Research Part A: Policy and Practice*, 35(6):475 – 494.

[27] Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730.

[28] Chen, D., Wang, J., Zou, F., Hou, W., and Zhao, C. (2012). An improved group search optimizer with operation of quantum-behaved swarm and its application. *Applied Soft Computing Journal*, 12(2):712–725.

[29] Chen, L.-Q. (2002). Phase-field models for microstructure evolution. *Annual review of materials research*, 32(1):113–140.

[30] Chen, W.-N., Zhang, J., Lin, Y., Chen, N., Zhan, Z.-H., Chung, H.-H., Li, Y., and Shi, Y.-H. (2013). Particle swarm optimization with an aging leader and challengers. *IEEE Transactions on Evolutionary Computation*, 17(2):241–258.

[31] Christensen, P. W. and Klarbring, A. (2008). *An introduction to structural optimization*, volume 153. Springer Science and Business Media.

[32] Clark, P. (1989). *Exemplar-based reasoning in geological prospect appraisal*. Citeseer.

[33] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[34] Dantzig, G. B. (1990). Origins of the simplex method. *A History of Scientific Computing*, pages 141–151.

[35] Dapogny, C., Faure, A., Michailidis, G., Allaire, G., Couvelas, A., and Estevez, R. (2017). Geometric constraints for shape and topology optimization in architectural design. *Computational Mechanics*, 59(6):933–965.

[36] Das, S. and Suganthan, P. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31.

[37] Davidon, W. C. (1991). Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17.

[38] Deaton, J. D. and Grandhi, R. V. (2014). A survey of structural and multidisciplinary continuum topology optimization: post 2000. *Structural and Multidisciplinary Optimization*, 49(1):1–38.

[39] Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387.

[40] Derrac, J., Garcia, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

[41] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655.

[42] Dong, C., Loy, C. C., He, K., and Tang, X. (2016a). Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307.

[43] Dong, C., Loy, C. C., and Tang, X. (2016b). Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*, pages 391–407. Springer.

[44] Dorigo, M. and Stutzle, T. (2004). *Ant Colony Optimization*. The MIT Press.

[45] Duarte, L. S., Celes, W., Pereira, A., Menezes, I. F., and Paulino, G. H. (2015). Polytop++: an efficient alternative for serial and parallel topology optimization on cpus & gpus. *Structural and Multidisciplinary Optimization*, 52(5):845–859.

[46] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

[47] Eschenauer, H. A., Kobelev, V. V., and Schumacher, A. (1994). Bubble method for topology and shape optimization of structures. *Structural optimization*, 8(1):42–51.

[48] Fahlman, S. E., Hinton, G. E., and Sejnowski, T. J. (1983). Massively parallel architectures for al: Netl, thistle, and boltzmann machines. In *National Conference on Artificial Intelligence, AAAI*.

[49] Fischer, A. and Igel, C. (2012). An introduction to restricted boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*, pages 14–36. Springer.

[50] Fischer, A. and Igel, C. (2014). Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39.

[51] Fletcher, R. (1970). A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322.

[52] Fletcher, R. and Powell, M. J. (1963). A rapidly convergent descent method for minimization. *The computer journal*, 6(2):163–168.

[53] Floudas, C. A. (2000). *Deterministic Global Optimization: Theory, Methods and Applications*. Springer-Verlag, Boston, MA.

[54] Freund, Y. and Haussler, D. (1992). Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in neural information processing systems*, pages 912–919.

[55] Fried, I. (1969). Gradient methods for finite-element eigenproblems. *AIAA Journal*, 7(4):739–741.

[56] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130.

[57] Gandomi, A. and Alavi, A. (2012). Krill herd: A new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation*, 17(12):4831–4845.

[58] Gao, X., Li, L., and Ma, H. (2017). An adaptive continuation method for topology optimization of continuum structures considering buckling constraints. *International Journal of Applied Mechanics*, 9(07):1750092.

[59] Geem, Z. (2010). Recent advances in harmony search algorithm. *Studies in Computational Intelligence*.

[60] Geem, Z. W., Kim, J. H., and Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *SIMULATION*, 76(2):60–68.

[61] Gilmore, P. and Kelley, C. T. (1995). An implicit filtering algorithm for optimization of functions with many local minima. *SIAM J. Optim*, 5:269–285.

[62] Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26.

[63] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[64] Hager, W. W. and Zhang, H. (2006). A survey of nonlinear conjugate gradient methods. *Pacific journal of Optimization*, 2(1):35–58.

[65] Hennig, P. and Kiefel, M. (2012). Quasi-newton methods: A new direction. *Journal of Machine Learning Research*, 14:843–865.

[66] Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the theory of neural computation.* Addison-Wesley/Addison Wesley Longman.

[67] Hestenes, M. R. (1956). The conjugate gradient method for solving linear systems. In *Proc. Symp. Appl. Math VI, American Mathematical Society*, pages 83–102.

[68] Hestenes, M. R. and Stiefel, E. (1952). *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC.

[69] Hiller, J. and Lipson, H. (2012). Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466.

[70] Hinton, G. E. (1999). Products of experts. *IET*.

[71] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

[72] Hinton, G. E. (2007a). Boltzmann machine. *Scholarpedia*, 2(5):1668. revision #91076.

[73] Hinton, G. E. (2007b). Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434.

[74] Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer.

[75] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

[76] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[77] Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA.

[78] Hooke, R. and Jeeves, T. A. (1961). " direct search" solution of numerical and statistical problems. *J. ACM*, 8(2):212–229.

[79] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

[80] Hornby, G. S., Lipson, H., and Pollack, J. B. (2001). Evolution of generative design systems for modular physical robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 4, pages 4146–4151. IEEE.

[81] Hu, Z., Bao, Y., and Xiong, T. (2014). Partial opposition-based adaptive differential evolution algorithms: Evaluation on the cec 2014 benchmark set for real-parameter optimization. *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, pages 2259–2265.

[82] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154.

[83] Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243.

[84] Hunter, W. et al. (2017). Topy - topology optimization with python. https://github.com/williamhunter/topy.

[85] Irani, M. and Peleg, S. (1991). Improving resolution by image registration. *CVGIP: Graphical models and image processing*, 53(3):231–239.

[86] Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181.

[87] Kallioras, N., Lagaros, N., and Avtzis, D. (2016). Pity beetle algorithm - a new metaheuristic inspired by bark beetles for solving engineering problemsk. In *11th HSTAM International Congress on Mechanics*.

[88] Kallioras, N., Lagaros, N., Karlaftis, M., and Pachy, P. (2014a). Harmony search design optimization of onshore wind farms. In *First International Conference on Engineering and Applied Sciences Optimization Opt-i*.

[89] Kallioras, N., Piliounis, G., Karlaftis, M. G., and Lagaros, N. D. (2013). Scheduling transportation networks and reliability analysis of geostructures using metaheuristics. In *Metaheuristics in Water, Geotechnical and Transport Engineering*, pages 345–363. Elsevier.

[90] Kallioras, N. A., Kazakis, G., and Lagaros, N. (2018a). Deep learning assisted topology optimization. In *The Tenth International Conference on Engineering Computational Technology*.

[91] Kallioras, N. A., Kepaptsoglou, K., and Lagaros, N. D. (2015a). Transit stop inspection and maintenance scheduling: A gpu accelerated metaheuristics approach. *Transportation Research Part C: Emerging Technologies*, 55:246 – 260. Engineering and Applied Sciences Optimization (OPT-i) - Professor Matthew G. Karlaftis Memorial Issue.

[92] Kallioras, N. A. and Lagaros, N. D. (2017). A real-time emergency inspection scheduling tool following a seismic event. In *Computational Methods in Earthquake Engineering*, pages 405–418. Springer.

[93] Kallioras, N. A., Lagaros, N. D., and Avtzis, D. N. (2018b). Pity beetle algorithm – a new metaheuristic inspired by the behavior of bark beetles. *Advances in Engineering Software*, 121:147 – 166.

[94] Kallioras, N. A., Lagaros, N. D., and Karlaftis, M. G. (2014b). An improved harmony search algorithm for emergency inspection scheduling. *Engineering Optimization*, 46(11):1570–1592.

[95] Kallioras, N. A., Lagaros, N. D., Karlaftis, M. G., and Pachy, P. (2015b). Optimum layout design of onshore wind farms considering stochastic loading. *Advances in Engineering Software*, 88:8 – 20.

[96] Kang, F., Li, J., and Ma, Z. (2011). Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. *Information Sciences*, 181(16):3508–3531.

[97] Karaboga, D. and Basturk, B. (2008). On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing Journal*, 8(1):687–697.

[98] Karakasis, M. K., Giotis, A. P., and Giannakoglou, K. C. (2003). Inexact information aided, low-cost, distributed genetic algorithms for aerodynamic shape optimization. *International Journal for Numerical Methods in Fluids*, 43(10-11):1149–1166.

[99] Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints. Master's thesis, Department of Mathematics, Univ. of Chicago, Illinois.

[100] Kashan, H. (2015). A new metaheuristic for optimization: Optics inspired optimization (oio). *Computers and Operations Research*, 55:99–125.

[101] Kazakis, G., Kanellopoulos, I., Sotiropoulos, S., and Lagaros, N. D. (2017). Topology optimization aided structural design: Interpretation, computational aspects and 3d printing. *Heliyon*, 3(10):e00431.

[102] Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Series in Evolutionary Computation*. Morgan Kaufmann Publishers.

[103] Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948.

[104] Kennedy, J. and Mendes, R. (2002). Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1671–1676.

[105] Kim, J., Kwon Lee, J., and Mu Lee, K. (2016). Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654.

[106] Knizek, M., Stauffer, C., Avtzis, D., and Wegensteiner, R. (2005). *Pityogenes chalcographus*. Forestry Compendium.

[107] Kolda, T. G., Lewis, R. M., and Torczon, V. (2003). Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482.

[108] Konstantinidou, M., Kallioras, N., Lagaros, N., Kepaptsoglou, K., and Karlaftis, M. (2014). Planning post-disaster operations in a high-way network. In *First International Conference on Engineering and Applied Sciences Optimization Opt-i*.

[109] Koumousis, V. K. and Katsaras, C. P. (2006). A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10(1):19–28.

[110] Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V., and Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17.

[111] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[112] Kuhn, H. and Tucker, A. (1950). Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492.

[113] Lagaros, N., Plevris, V., and Papadrakakis, M. (2005). Multi-objective design optimization using cascade evolutionary computations. *Computer Methods in Applied Mechanics and Engineering*, 194:3496–3515.

[114] Lagaros, N. D., Papadrakakis, M., and Kokossalakis, G. (2002). Structural optimization using evolutionary algorithms. *Computers and Structures*, 80(7):571 – 589.

[115] Lagaros, N. D., Vasileiou, N., and Kazakis, G. (2018). A *c#* code for solving 3d topology optimization problems using sap2000. *Optimization and Engineering*.

[116] Le Roux, N. and Bengio, Y. (2008). Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649.

[117] LeCun, Y., Bengio, Y., and Hinton, G. (2015a). Deep learning. *nature*, 521(7553):436.

[118] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[119] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[120] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

[121] LeCun, Y. et al. (2015b). Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20.

[122] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017a). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690.

[123] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017b). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690.

[124] Li, L. and Khandelwal, K. (2015). Volume preserving projection filters and continuation methods in topology optimization. *Engineering Structures*, 85:144–161.

[125] Li, X. and Yao, X. (2012). Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224.

[126] Liang, J., Qin, A., Suganthan, P., and Baskar, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, 10(3):281–295.

[127] Liang, J., Qu, B.-Y., and Suganthan, P. (2013a). *Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, Technical Report 201311*. Nanyang Technological University.

[128] Liang, J., Qu, B.-Y., Suganthan, P., and Hernández-Díaz, A. (2013b). *Problem definitions and evaluation criteria for the CEC 2013 special session and competition on real-parameter optimization, Technical Report 201212*. Nanyang Technological University.

[129] Liang, J. and Suganthan, P. (2005). Dynamic multi-swarm particle swarm optimizer. In *Proceedings of IEEE Swarm Intelligence Symposium SIS 2005*, pages 124–129.

[130] Lindenmayer, A. (1975). Developmental algorithms for multicellular organisms: A survey of l-systems. *Journal of Theoretical Biology*, 54(1):3–22.

[131] LISA-Lab (2014). Tutorial, deep learning. *University of Montreal*.

[132] Loshchilov, I., Stuetzle, T., and Liao, T. (2013). Ranking results of cec'13 special session and competition on real-parameter single objective optimization. In *IEEE Congress on Evolutionary Computation (CEC)*.

[133] Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.-Y., et al. (2015). Traffic flow prediction with big data: A deep learning approach. *IEEE Trans. Intelligent Transportation Systems*, 16(2):865–873.

[134] Maaranen, H., Miettinen, K., and Penttinen, A. (2007). On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37:405–436.

[135] Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

[136] Marsden, A. L., Feinstein, J. A., and Taylor, C. A. (2008). A computational framework for derivative-free optimization of cardiovascular geometries. *Computer Methods in Applied Mechanics and Engineering*, 197(21):1890 – 1905.

[137] Marsland, S. (2011). *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC.

[138] Martínez-Frutos, J. and Herrero-Pérez, D. (2016). Large-scale robust topology optimization using multi-gpu systems. *Computer Methods in Applied Mechanics and Engineering*, 311:393–414.

[139] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[140] Meinhardt, H. (1976). Morphogenesis of lines and nets. *Differentiation*, 6(2):117–123.

[141] Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210.

[142] Meza, J. C. (2010). Steepest descent. *WIREs Comput. Stat.*, 2(6):719–722.

[143] Michalewicz, Z. (2012). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

[144] Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N. (1986). The multi-purpose incremental learning system aq15 and its testing application to three medical domains. *Proc. AAAI 1986*, pages 1–041.

[145] Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89:228–249.

[146] Mitchell, T. M. (1997). *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill.

[147] Mlejnek, H. (1992). Some aspects of the genesis of structures. *Structural optimization*, 5(1-2):64–69.

[148] N. S. Khot, L. Berke; Venkayya, V. B. (1979). Comparison of optimality criteria algorithms for minimum weight design of structures. *AIAA Journal*, 17.

[149] Nassirtoussi, A. K., Aghabozorgi, S., Wah, T. Y., and Ngo, D. C. L. (2015). Text mining of news-headlines for forex market prediction: A multi-layer dimension reduction algorithm with semantics and sentiment. *Expert Systems with Applications*, 42(1):306–324.

[150] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

[151] Novak, V. (1976). *Atlas of Insects Harmful to Forest Trees*. Elsevier Scientific Publishing Company.

[152] Olsson, A., Sandberg, G., and Dahlblom, O. (2003). On latin hypercube sampling for structural reliability analysis. *Structural Safety*, 25:47–68.

[153] Papoutsis-Kiachagias, E. M. and Giannakoglou, K. C. (2016). Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: Industrial applications. *Archives of Computational Methods in Engineering*, 23(2):255–299.

[154] Park, S. C., Park, M. K., and Kang, M. G. (2003). Super-resolution image reconstruction: a technical overview. *IEEE signal processing magazine*, 20(3):21–36.

[155] Parsopoulos, K. and Vrahatis, M. (2004). Upso: A unified particle swarm optimization scheme. *Lecture Series on Computational Sciences*, pages 868–873.

[156] Patterson, J. and Gibson, A. (2017). *Deep Learning: A Practitioner's Approach*. " O'Reilly Media, Inc.".

[157] Pedro, H.-T. C. and Kobayashi, M. H. (2011). On a cellular division method for topology optimization. *International Journal for Numerical Methods in Engineering*, 88(11):1175–1197.

[158] Peram, T., Veeramachaneni, K., and Mohan, C. (2003). Fitness-distance-ratio based particle swarm optimization. In *Proceedings of IEEE Swarm Intelligence Symposium SIS 2003*, pages 124–129.

[159] Pfeffer, A. (1995). *Zentral und westpalaarktische Borken und Kernkafer*. Pro Entomologia, c/o Naturhistorisches Museum Basel.

[160] Pirlot, M. (1996). General local search methods. *European Journal of Operational Research*, 92(3):493 – 511.

[161] Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57.

[162] Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105.

[163] Postner, M. (1974). *Scolytidae, Borkenkafer. In: Die Forstschädlinge Europas. Schwenke W*. P. Parey.

[164] Poultney, C., Chopra, S., Cun, Y. L., et al. (2007). Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144.

[165] Powell, M. J. D. (1994). *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*. Springer Netherlands, Dordrecht.

[166] Pradhan, B. (2013). A comparative study on the predictive ability of the decision tree, support vector machine and neuro-fuzzy models in landslide susceptibility mapping using gis. *Computers and Geosciences*, 51:350–365.

[167] Preux, P., Munos, R., and Valko, M. (2014). Bandits attack function optimization. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, pages 2245–2252.

[168] Querin, O., Steven, G., and Xie, Y. (1998). Evolutionary structural optimisation (eso) using a bidirectional algorithm. *Engineering computations*, 15(8):1031–1048.

[169] Querin, O., Steven, G., and Xie, Y. (2000). Evolutionary structural optimisation using an additive algorithm. *Finite Elements in Analysis and Design*, 34(3-4):291–308.

[170] Rao, S. S. (2009). *Engineering Optimization: Theory and Practice: Fourth Edition*. John Wiley and Sons.

[171] Ratnaweera, A., Halgamuge, S., and Watson, H. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255.

[172] Riche, R. L. and Haftka, R. (2012). On global optimization articles in smo. *Structural and Multidiscipli-nary Optimization*, 46(5):627–629.

[173] Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

[174] Rojas-Labanda, S. and Stolpe, M. (2015). Automatic penalty continuation in structural topology optimization. *Structural and Multidisciplinary Optimization*, 52(6):1205–1221.

[175] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

[176] Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition. volume 1. foundations*. MIT Press, Cambridge, Ma.

[177] Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

[178] Sahinidis, N. V. (1996). Baron: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205.

[179] Salakhutdinov, R. and Hinton, G. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419.

[180] Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.

[181] Salakhutdinov, R. and Hinton, G. (2012). An efficient learning procedure for deep boltzmann machines. *Neural Computation*, 24(8):1967–2006.

[182] Salomon, R. (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278.

[183] Sarker, R., Elsayed, S., and Ray, T. (2004). Differential evolution with dynamic parameters selection for optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(5):689–707.

[184] Schmidhuber, J. (1993). Habilitation thesis: Netzwerkarchitekturen, zielfunktionen und kettenregel.

[185] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[186] Schwerdtfeger, F. (1929). Ein beitrag zur fortpflanzungsbiologie des borkenkafers pityogenes chalcographus. *L. Zeitschrift für angewandte Entomologie*, 15:335–427.

[187] Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656.

[188] Shawe-Taylor, J., Cristianini, N., et al. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

[189] Shi, Y. and Eberhart, R. (1998). Modified particle swarm optimizer. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 69–73.

[190] Shi, Y. and Eberhart, R. (2001). Particle swarm optimization with fuzzy adaptive inertia weight. In *Proceedings of the Workshop on Particle Swarm Optimization*, pages 101–106.

[191] Shi, Y., Pun, C., Hu, H., and Gao, H. (2016). An improved artificial bee colony and its application. *Knowledge-Based Systems*, 107:14–31.

[192] Sigmund, O. (1994). *Design of material structures using topology optimization*. PhD thesis, Technical University of Denmark Denmark.

[193] Sigmund, O. (2001). A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127.

[194] Sigmund, O. and Maute, K. (2013). Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055.

[195] Singh, V. and Gu, N. (2012). Towards an integrated generative design framework. *Design studies*, 33(2):185–207.

[196] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE.

[197] Sommer, R. and Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE.

[198] Sosnovik, I. and Oseledets, I. (2017). Neural networks for topology optimization. *arXiv preprint arXiv:1709.09578*.

[199] Stiny, G. (1980). Introduction to shape and shape grammars. *Environment and planning B: planning and design*, 7(3):343–351.

[200] Stockman, G. and Shapiro, L. G. (2001). *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[201] Storn, R. and Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359.

[202] Svanberg, K. (1987). The method of moving asymptotes - a new method for structural optimization. *Optimization and Systems Theory*, 24(2):359–373.

[203] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[204] Talbi, E. (2006). *Metaheuristics: From Design to Implementation*. John Wiley and Sons.

[205] Talischi, C., Paulino, G. H., Pereira, A., and Menezes, I. F. (2012a). Polymesher: a general-purpose mesh generator for polygonal elements written in matlab. *Structural and Multidisciplinary Optimization*, 45(3):309–328.

[206] Talischi, C., Paulino, G. H., Pereira, A., and Menezes, I. F. (2012b). Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*, 45(3):329–357.

[207] Tamilselvan, P. and Wang, P. (2013). Failure diagnosis using deep belief learning based health state classification. *Reliability Engineering & System Safety*, 115:124–135.

[208] Tanabe, R. and Fukunaga, A. (2014). Improving the search performance of shade using linear population size reduction. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, pages 1658–1665.

[209] Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM J. on Optimization*, 7(1):1–25.

[210] van den Bergh, F. and Engelbrecht, A. (2004). A cooperative approach to participle swam optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239.

[211] van Laarhoven, P. and Aarts, E. (2010). *Simulated Annealing: Theory and Applications (Mathematics and Its Applications)*. Kluwer Academic Publishers.

[212] Villaggi, L., Stoddart, J., Nagy, D., and Benjamin, D. (2018). Survey-based simulation of user satisfaction for generative design in architecture. In *Humanizing Digital Reality*, pages 417–430. Springer.

[213] Vité, J. (1965). Ist die vorbeugende begiftung von fangbäumen zweckmässig? *Allgemeine Forstzeitschrift für Waldwirtschaft and Umweltvorsorge*, 20:438–439.

[214] Von Neumann, J. et al. (1951). The general and logical theory of automata. *1951*, pages 1–41.

[215] Wang, M. Y., Wang, X., and Guo, D. (2003). A level set method for structural topology optimization. *Computer methods in applied mechanics and engineering*, 192(1-2):227–246.

[216] Wang, X., Xu, S., Zhou, S., Xu, W., Leary, M., Choong, P., Qian, M., Brandt, M., and Xie, Y. M. (2016). Topological design and additive manufacturing of porous metals for bone scaffolds and orthopaedic implants: a review. *Biomaterials*, 83:127–141.

[217] Welling, M., Rosen-Zvi, M., and Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems*, pages 1481–1488.

[218] Wilson, D. R. and Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286.

[219] Wolfram, S. (2002). *A new kind of science*, volume 5. Wolfram media Champaign, IL.

[220] Wood, S. and Bright, D. (1992). A catalog of scolytidae and platypodidae (coleoptera). *Great Basin Naturalist Memoirs*, 13:1–833, 835–1553.

[221] Xie, Y. M. and Steven, G. P. (1993). A simple evolutionary procedure for structural optimization. *Computers & structures*, 49(5):885–896.

[222] Yamashita, T., Tanaka, M., Yoshida, E., Yamauchi, Y., and Fujiyoshii, H. (2014). To be bernoulli or to be gaussian, for a restricted boltzmann machine. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 1520–1525. IEEE.

[223] Yang, J., Wright, J., Huang, T. S., and Ma, Y. (2010). Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873.

[224] Yang, X. (2008). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press.

[225] Yu, C., Kelley, L., Zheng, S., and Tan, Y. (2014). Fireworks algorithm with differential mutation for solving the cec 2014 competition problems. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, pages 3238–3245.

[226] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

[227] Zheng, Y. (2015). Water wave optimization: A new nature-inspired metaheuristic. *Computers and Operations Research*, 55:1–11.

[228] Zhigljavsky, A. and Žilinskas, A. (2008). *Stochastic Global Optimization*. Springer, Boston, MA.

[229] Zhou, M. and Rozvany, G. (1991). The coc algorithm, part ii: topological, geometrical and generalized shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 89(1-3):309–336.

[230] Zhu, J.-H., Zhang, W.-H., and Xia, L. (2016). Topology optimization in aircraft and aerospace structures design. *Archives of Computational Methods in Engineering*, 23(4):595–622.

[231] Zuber, M. (1994). Ökologie der borkenkäfer. *Biologie in unserer Zeit*, 3:144–152.

[232] Zumr, V. and Zoldán, T. (1981). Reproductive cycles of ips typographus, i. amitinus and pityogenes chalcographus (coleoptera, scolytidae). *Acta Entomologia Bohemoslovaca*, 78:280–289.