

Weitere Algorithmenmuster

- Randomisierte und heuristische Approximationsverfahren
 - Las-Vegas-Algorithmen
 - Monte-Carlo-Algorithmen
 - Genetische / evolutionäre Algorithmen
 - Simulated Annealing
- Dynamische Programmierung

Randomisierte / heuristische Approximationsverfahren

Randomisierte Algorithmen:

- Ein randomisierter Algorithmus versucht, durch die Wahl von zufälligen Zwischenergebnissen zu einem (näherungsweise) korrekten Ergebnis zu gelangen.
- Man kann den Erwartungswert der Rechenzeit und/oder die Fehler- bzw. Versagenswahrscheinlichkeit abschätzen.
- Nicht deterministisch.
- Randomisierte Algorithmen sind in vielen Fällen einfacher zu verstehen, einfacher zu implementieren und effizienter als deterministische Algorithmen

Heuristische Algorithmen:

- Algorithmen, bei denen nur intuitiv plausibel ist, dass sie gute Ergebnisse liefern, oder Algorithmen, bei denen man dies durch Experimente auf typischen Eingaben bewiesen hat, bezeichnet man dagegen als heuristische Algorithmen.

Randomisierte / heuristische Approximationsverfahren

Randomisierte Verfahren:

- Las-Vegas-Algorithmen
- Monte-Carlo-Algorithmen

Heuristische Verfahren:

- Genetische / evolutionäre Algorithmen
- Simulated Annealing, andere Namen:
 - Statistical Cooling
 - Monte Carlo Annealing
 - Probabilistic Hill Climbing
 - Stochastic Relaxation
 - Probabilistic Exchange Algorithm
- etc.

Las-Vegas-Algorithmen

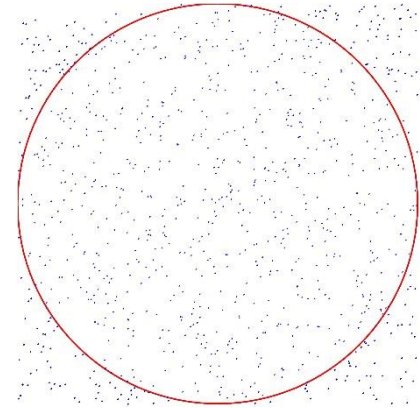
- Liefern nie ein falsches Ergebnis
- allenfalls wird keine Lösung gefunden
- Beispiel: Variante von Quicksort, Pivotelement zufällig ausgewählt. Der Erwartungswert der Laufzeit des randomisierten Quicksort-Algorithmus für n Elemente ist $O(n \log n)$

Monte-Carlo-Algorithmen

- Liefern ein mehr oder weniger gutes Ergebnis.
- Die Qualität von Monte-Carlo-Algorithmen kann man durch eine obere Schranke für die Fehlerwahrscheinlichkeit beschreiben.

Wir möchten Pi berechnen.

Wir können eine Simulation schreiben, die 100.000 Punkte zufällig einem Quadrat platziert. Pi ergibt sich aus dem Verhältnis der Punkt in dem Kreis zur Gesamtzahl der Punkte.

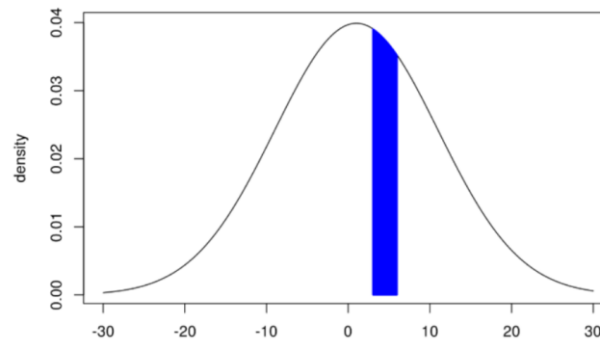


```
int nSuccess = 0;
double x, y;

for (int i = 0; i < 1000000 ; i++) {
    x = Math.random
    y = Math.random();
    if ( x*x + y*y <= 1 ) nSuccess++;
}
System.out.println(4*(double)nSuccess/(double) 1000000);
```

Angenommen, wir haben eine Instanz einer Normalverteilung mit einem Mittelwert von 1 und einer Standardabweichung von 10. Wir wollen das Integral von 3 bis 6 finden:

$$\int_3^6 \frac{1}{10\sqrt{2\pi}} e^{-\frac{(x-1)^2}{2 \cdot 10^2}} dx$$



Wir können eine Simulation schreiben, die 100.000 Proben aus dieser Verteilung entnimmt und sehen, wie viele Werte zwischen 3 und 6 liegen.

Das Ergebnis, das wir erhalten haben, ist: 0,1122, was nicht allzu weit von 0,112203 entfernt ist.

Idee (Evolution nachbilden):

- Eine vereinfachte Vorstellung der Evolution wird in der Informatik idealisiert und künstlich im Computer nachgebildet.
- Die Güte eines Lösungskandidaten wird explizit mit einer Fitnessfunktion berechnet, sodass verschiedene Kandidaten vergleichbar sind.

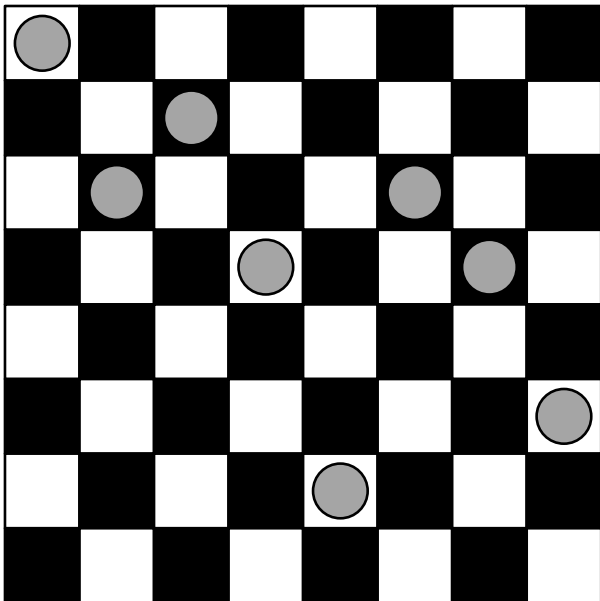


Ablauf:

- Initialisierung: Die erste Generation von Lösungskandidaten wird (meist zufällig) erzeugt.
- Durchlaufe die folgenden Schritte, bis ein Abbruchkriterium erfüllt ist:
 1. **Evaluation Fitness**: Jedem Lösungskandidaten der Generation wird entsprechend seiner Güte ein Wert der Fitnessfunktion zugewiesen.
Abbruch?
 2. **Selektion**: Auswahl der neuen Generation und Auswahl von Individuen für die Rekombination
 3. **Rekombination**: Kombination der ausgewählten Individuen
 4. **Mutation**: Zufällige Veränderung der Nachfahren

Beispiel: n-Damen auf Schachbrett

Speicherung: Folge von Zahlen, i. Zahl: Position von Dame in Spalte i (hatten wir schon mal, z.B. [1, 3, 2, 4, 7, 3, 4, 6]):



Fitness:

Anzahl nicht bedrohender Damenpaare:

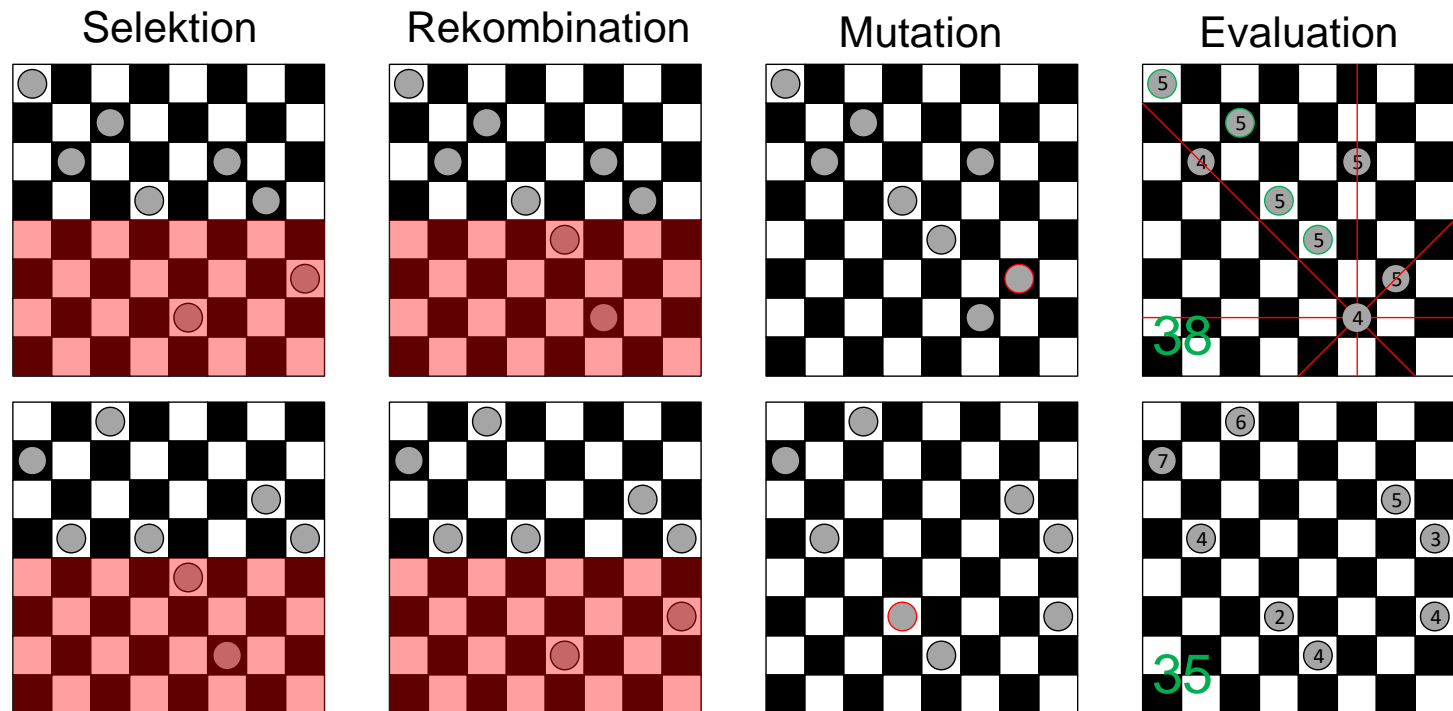
- schlechterster Wert: 0, jede Dame bedroht jede andere
- bester Wert: $n*(n-1)/2$

Mutation:

- Ändere einen Wert aus $\{1, \dots, n\}$ zufällig
- entspricht: bewege Dame in einer Zeile auf anderen Platz

Beispiel: n-Damen auf Schachbrett

Auf den besten Individuen werden mittels Rekombination und Mutatation neue (zusätzliche) Nachfahren erzeugt:



Idee (Simulated Annealing = Simuliertes Glühen):

- Allgemeines Verfahren zur Lösung kombinatorischer Optimierungsprobleme
- der Algorithmus basiert auf der Simulation eines in der Natur vorkommenden Prozesses, des Abkühlens von Stoffen.
- Metropolis [1953] Untersuchung von Vielkörpersystemen, Fluiden, Gasen, Festkörper auf Mittelwerte, Abweichungen, Temp., Druck
 - abkühlende Flüssigkeiten streben beim Abkühlen nach einer minimalen Energiebilanz E (stabile, regelmäßige Struktur), nur möglich bei langsamer Kühlung
 - ist Kühlung zu schnell, Teilchen ordnen sich unregelmäßig an schlechte Energiebilanz (Auftreten von Nebenminima)

Prinzip:

- Start with initial configuration
- Repeatedly search neighborhood and select a neighbor as candidate
- Evaluate some cost function (or fitness function) and accept candidate if "better"; if not, select another neighbor
- Stop if quality is sufficiently high, if no improvement can be found or after some fixed time

Needed are:

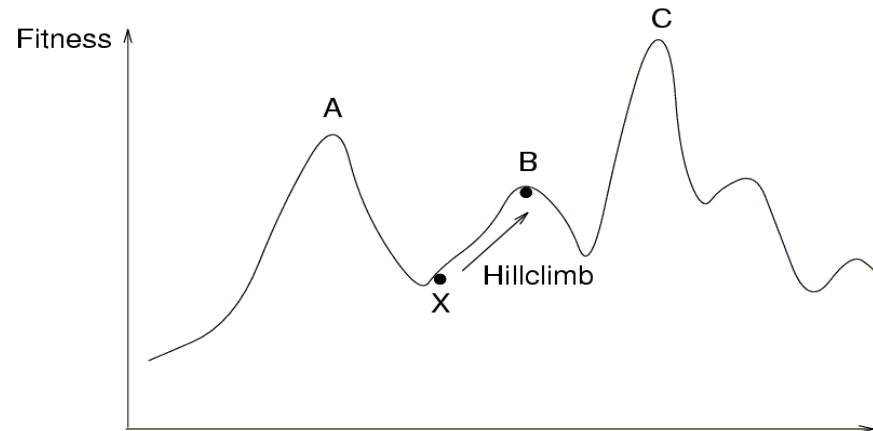
- A method to generate initial configuration
- A transition or generation function to find a neighbor as next candidate
- A cost function
- An Evaluation Criterion
- A Stop Criterion

Performance, simple Iterative Improvement or Hill Climbing:

- Candidate is always and only accepted if cost is lower (or fitness is higher) than current configuration
- Stop when no neighbor with lower cost (higher fitness) can be found

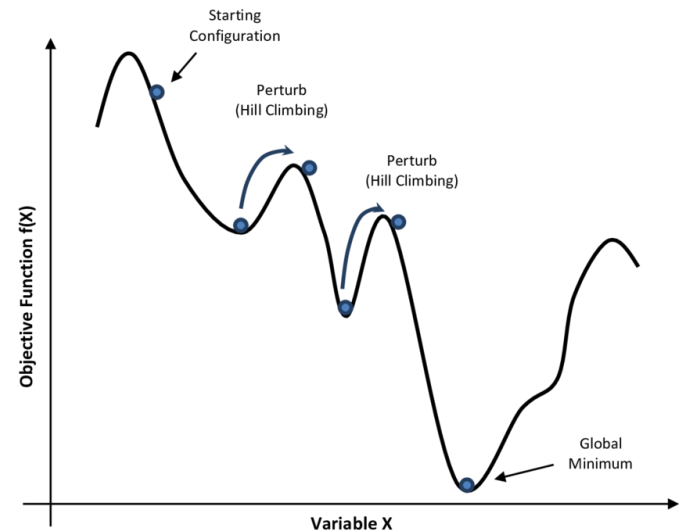
Disadvantages:

- Local optimum as best result
- Local optimum depends on initial configuration
- Generally no upper bound on iteration length



How to cope with disadvantages

- Repeat algorithm many times with different initial configurations
- Use information gathered in previous runs
- Use a more complex Generation Function to jump out of local optimum
- Use a more complex Evaluation Criterion that accepts sometimes (randomly) also solutions away from the (local) optimum
 - Do sometimes accept candidates with higher cost to escape from local optimum
 - Adapt the parameters of this Evaluation Function during execution

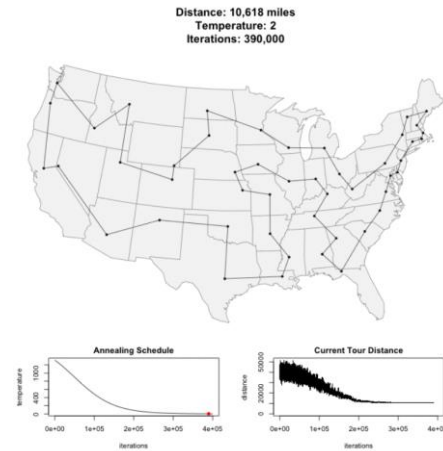
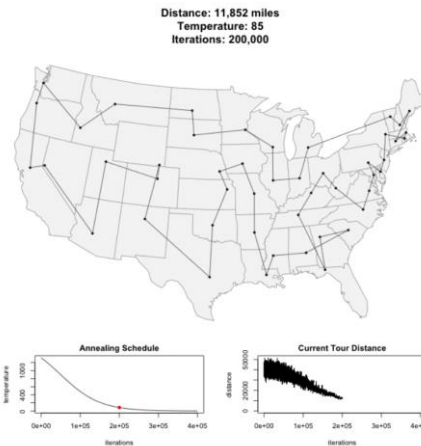


- SA is a general solution method that is easily applicable to a large number of problems
- "Tuning" of the parameters (initial c , decrement of c , stop criterion) is relatively easy – hard to find optimum
- Generally the quality of the results of SA is good, although it can take a lot of time
- Results are generally not reproducible: another run can give a different result
- SA can leave an optimal solution and not find it again (so try to remember the best solution found so far)
- Proven to find the optimum under certain conditions; one of these conditions is that you must run forever

Salesman Travelling Problem:

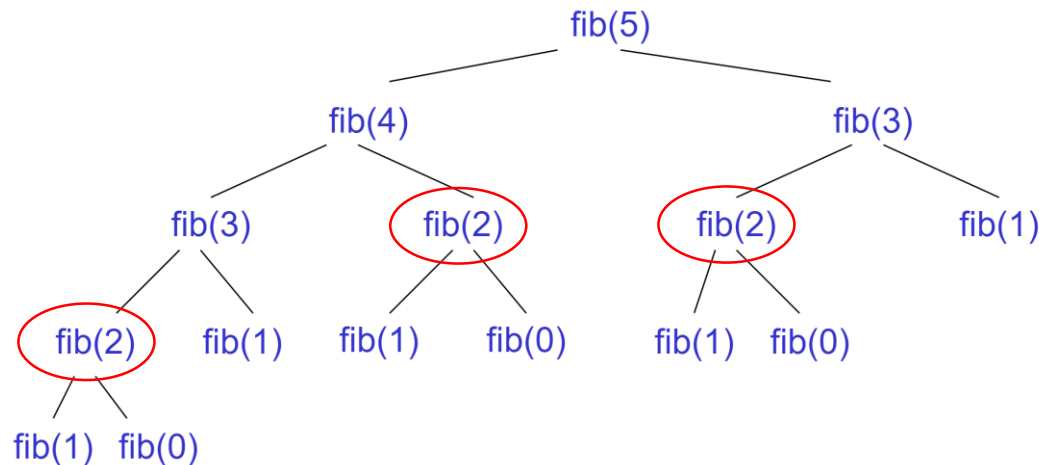
1. Start with random tour
2. Pick a new candidate tour at random.
One way to pick a new tour is to choose two cities on the tour randomly, and then reverse the portion of the tour that lies between them.
3. If the candidate tour is better, accept it as the new tour.
4. If the candidate tour is worse, still maybe accept it, according to some probability:
The probability of accepting an inferior tour is a function of how much longer the candidate is, and the "temperature of the annealing process". A higher temperature makes you more likely to accept an inferior tour.
5. Go back to step 2 and repeat many times, lowering the temperature a bit at each iteration, until you get to a low temperature and arrive at your (hopefully global, possibly local) minimum.

Simulated Annealing



<http://toddschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/>





- Idee: Optimierungsproblem durch Aufteilung in Teilprobleme lösen und Zwischenresultaten systematischen speichern.
- Beispiel: Berechnung Fibonacci-Zahlen



-> fib(2) als Zwischenresultat speichern und nur einmal berechnen.

Dynamische Programmierung

Beispiel: Rucksackproblem

1.	2.	3.	4.	5.
				
Volumen: 1l	Volumen: 2l	Volumen: 9l	Volumen: 8l	Volumen: 9l
Wert: 2'000	Wert: 3'000	Wert: 10'000	Wert: 11'000	Wert: 17'000

Wir hatten das Problem mit Backtracking gelöst. Wie können wir diesen Ansatz mittels dynamischer Programmierung verbessern?

Wir hatten das Problem mit Backtracking gelöst. Wie können wir diesen Ansatz mittels dynamischer Programmierung verbessern?

- Haben wir schon 3 Gegenstände ausprobiert, und im Baum haben zwei Knoten jetzt dasselbe Restgewicht, dann muss nur einer dieser Pfade berechnet werden, um die möglichen Optima zu berechnen.

Beispiel:

- Wir haben die Varianten zu den Gegenständen 3 bis 5 gebildet, dann haben wir zwei Varianten mit dem selben Restgewicht (3 und 4, sowie 4 und 5, je 17kg).
- Es genügt daher, die optimale Variante mit den Gegenständen 1 und 2 (mit dem gegebenen Restgewicht) zu berechnen und dieses Resultat in beiden Fällen wiederzuverwenden.

Was es alles nicht in diese Vorlesung geschafft hat:





























- Approximationsalgorithmen
- Parallele Algorithmen
- Lineare Programmierung
- Fuzzy Logic
- Geometrische Algorithmen
- Signalverarbeitung mit FFT
- Map-Reduce Framework
- Numerische Programmierung
- Bildverarbeitung
- Machine Learning
- Etc.

- Wofür verwendet man die O-Notation?
- Was ist der Unterschied zwischen Average und Worst Case Laufzeit?
- Nennen Sie einige typische Komplexitätsklassen
- Wie ist die Worst-Case-Laufzeit von binärer Suche?
- Erklären Sie wie binäre Suche funktioniert.
- Welche Laufzeit hat das Einfügen eines Elements an der ersten Position in einer einfach-verketteten Liste?
- Welche Laufzeit hat das Einfügen eines Elements an der ersten Position in einer ArrayList?
- Was ist der Unterschied zwischen einer einfach und einer doppelt verketteten Liste?
- Warum ist ein "tail"-pointer bei Listen oft hilfreich?
- Wie kann man in einer einfach verketteten Liste ein Element an Position i einfügen?
- Welche Laufzeit hat das Löschen eines Elementes in einer doppelt verketteten Liste?
- Geben Sie ein Beispiel für einen Greedy-Algorithmus
- ...

Prüfung

- Gemäss Doodle-Umfrage
- Raum dem Stundenplan entnehmen



	Jan 16 MI 08:00 08:15	Jan 16 MI 08:20 08:35	Jan 16 MI 08:40 08:55	Jan 16 MI 09:20 09:35	Jan 16 MI 09:40 09:55	Jan 16 MI 10:00 10:15	Jan 16 MI 10:40 10:55	Jan 16 MI 11:00 11:15	Jan 16 MI 11:20 11:35	Jan 16 MI 11:40 11:55	Jan 16 MI 13:00 13:15	Jan 16 MI 13:20 13:35	Jan 16 MI 13:40 13:55	Jan 16 MI 14:00 14:15	Jan 16 MI 14:20 14:35	Jan 16 MI 14:40 14:55	Jan 16 MI 15:00 15:15	Jan 17 DO 08:00 08:15	Jan 17 DO 08:20 08:35	Jan 17 DO 08:40 08:55	Jan 17 DO 09:20 09:35	Jan 17 DO 09:40 09:55	Jan 17 DO 10:00 10:15	Jan 17 DO 10:40 10:55	Jan 17 DO 11:00 11:15	Jan 17 DO 11:20 11:35	Jan 17 DO 11:40 11:55	Jan 17 DO 13:00 13:15	Jan 17 DO 13:20 13:35
27 Teilnehmer	✓ 0/1	✓ 0/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1	✓ 1/1
<div> Dein Name</div>	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
<div> Gian Hellinger</div>												✓																	
<div> kasikfi</div>													✓																
<div> Michèle Trebo</div>								✓																					
<div> Robin Frehner</div>														✓															
<div> Gabriel Koch</div>											✓																		
<div> Dennis Gehrig</div>																		✓											
<div> Yves Kaufmann</div>																			✓										
<div> Akex</div>																				✓									
<div> kopczynski, lorenz</div>															✓														
<div> Daniel Lerch</div>																													
<div> Ferenc Kuntic</div>												✓																	
<div> Raphael Mailänder</div>				✓																									
<div> Michael Schaufelbe...</div>					✓																								
<div> Nick Duong</div>																								✓					
<div> Mattia Ninivaggi</div>																		✓											
<div> Pascal Fivian</div>																												✓	
<div> Davide Giunta</div>										✓																			
<div> André Livramento</div>																													✓
<div> Cyril Schuhmacher</div>																									✓				
<div> Pascal Haupt</div>																				✓									
<div> Raphael Caradonna</div>			✓																										
<div> Dominique Reiser</div>											✓																		
<div> Dirk Furrer</div>																									✓				
<div> Simon Stucki</div>							✓																						
<div> Pascal Thalong</div>																													✓
<div> Marc Berli</div>																											✓		
<div> Ioannis Vettas</div>						✓																							