**Moving-target-TSP**

* Ziel: alle Knoten des Graphens besuchen
* Erweiterung des TSP´s: Die zu besuchenden Ziele sind in Bewegung -> Moving-target-TSP
* Gegeben wird eine Eingabe von Zielen S = {s\_1,...,s\_n}
* Jedes s\_1 bewegt sich mit einer konstanten Geschwindigkeit v\_i
* Der Verfolger startet im Ursprung mit der Maximalgeschwindigkeit v > |v\_i|
* Zeitkomplexität kann nicht besser als der Faktor 2^(Omega(Wurzel(n))) mal die optimale Lösung sein
* MTSP ist NP hart
* Betrachten nun im Folgenden zwei Varianten: Ziele befinden sich auf einer Linie (eindimensional); Anzahl der beweglichen Ziele ist klein
* **Lemma 1**: In jeder optimalen MTSP Route bewegt sich der Verfolger immer mit der Maximalgeschwindigkeit
* Mit den Bedingungen der letzten 2 Stichpunkte erarbeiten wir nun ein Alg., basierend auf dynamischer Programmierung, mit einer Laufzeit von O(n^2) heraus
* Dafür berechnen wir erst die Kosten alle Ziele links vom Ursprung zu besuchen und dann die auf der rechten Seite
* Anschließend berechnen wir dasselbe, nur spiegelverkehrt
* Wähle die Tour mit den geringsten Kosten
* Problem: ein unbounded error könnte auftreten. Bsp: 2 Ziele auf jeder Seite, jeweils ein schnelles und ein extrem langsames. Alle bewegen sich vom Ursprung weg. Mit dem gerade genannten Ansatz müsste jeweils ein schneller eingeholt werden, anstatt die schnellen Knoten als erstes zu besuchen
* **Lemma 2**: In einer optimalen Tour für 1dimensionale MTSP kann der Verfolger nicht die Richtung ändern, bis das schnellste Ziel in der Richtung besucht wurde
* Um nun eine Tour zu bestimmen müssen wir also nur die Knoten wissen, an denen der Verfolger die Richtung wechselt. Wir müssen also die Tour als eine Sequenz von Momentaufnahmen von den Drehpunkten betrachten und bezeichnen dies als Zustand
* Zustand = (s\_k, s\_f) – s\_k ist der gerade besuchte Knoten, s\_f ist der schnellste Knoten auf der anderen Seite
* Jede Tour hat denselben Start- und Endzustand (A\_0 und A\_final)
* Für jeden Zustand A\_i kann mit der Funktion t die kürzeste Zeit bis zum Erreichen des Zustandes berechnet werden. Es gilt t(A\_0) = 0
* Nach Lemma 2 gibt es immer bis zu zwei mögliche Übergänge von einem Zustand ausgehend, entweder auf der linken oder rechten Seite des Verfolgers
* Mit t(*tau*) wird die Zeit von einem Übergang zum nächsten Knoten bestimmt
* **Algorithmus:** 
  + Teilen die Ziele in zwei Listen auf: links und rechts vom Ursprung ausgesehen
  + Sortieren dann die Listen in absteigender Reihenfolge nach deren Geschwindigkeit
  + Durchlaufen die Listen und entfernen jedes Ziel, welches näher am Ursprung als andere Ziele liegt, die schneller sind
  + Nur in den übriggebliebenen Knoten wird der Verfolger potentiell seine Richtung ändern.
* Sei G = (V,E) ein Graph, dabei ist V die Menge der Zustände und E die Menge Übergänge zwischen den Zuständen. G ist azyklisch
* Der Verfolger kann nicht in den vorherigen Zustand zurückkehren, da kein Ziel ein zweites Mal besucht werden darf
* Der Algorithmus kann nun als Verallgemeinerung eines kürzester-Pfad-Problems im azyklischen Graphen G angesehen werden. Da Gewicht von *tau* hängt dabei vom Gewicht des kürzesten Pfads vom initialen Zustand A\_0 zum Startzustand A\_1 ab
* Die Laufzeit kann mit O(V+E) oder O(n^2) abgeschätzt werden
* Wenn wir Zustand A\_i erreichen, **eine** der drei Operationen durchgeführt:
  + Sofern A\_i keine Übergänge enthält, fahren wir mit dem nächsten Zustand in der Liste fort
  + Wenn der Verfolger alle Ziele einer Seite erreicht hat, erzeugen wir einen Übergang in den Endzustand A\_final
  + Wir erzeugen zwei Übergänge, welche den Verfolger entweder zum schnellsten Ziel auf der linken oder rechten Seite schickt
* Definieren nun die Zeit des Zustandes B mit t(B) = min{t(A) + t(*tau*) | *tau* : A -> B}. Dies ist die kürzeste Zeit von A\_0 bis B
* Nachdem wir alle Zustände besucht haben, können wir diese rückwärts von A\_final bis A\_0 durchgehen. Damit erhalten wir alle Zustände, in denen der Verfolger die Richtung in einer optimalen Tour ändert.
* Für jedes Paar der Wendepunkte erhalten wir eine Teilmenge der Ziele, die zwischen diesen erreicht werden.
* Sobald wir das Ziel in Teilmengen unterteilt haben, sortieren wir die Ziele in jeder Teilmenge nach ihren Abfangzeiten.
* Schließlich führen wir die sortierten Teilmengen zu einer kombinierten Abfangreihenfolge zusammen, um eine optimale Lösung zu erhalten.
* **Theorem 3**: Der Algorithmus für eindimensionale MTSP findet eine optimale Tour
* **Theorem 4**: Der Algorithmus für eindimensionale MTSP hat eine Laufzeit von O(n^2) bei n Zielen
* **Lemma 5**: Alle Übergänge können in konstanter, armortisierter Zeit pro Übergang durchlaufen werden
* Betrachten nun Instanzen von MTSP, in denen nur einige Ziele sich bewegen, während die Mehrheit eine feste Position hat. Damit kann ein effizienter Algorithmus mit der Leistungsgrenze von 1+*alpha* konstruiert werden, sofern die Anzahl an bewegenden Zielen ausreichend klein ist
* **Theorem 6**: MTSP mit maximal O(logn/(loglogn)) bewegenden Zielen kann in Polynomzeit mit der Leistungsgrenze 1 + *alpha* approximiert werden. Dabei ist *alpha* eine Leistungsgrenze einer beliebigen TSP-Heuristik

**MTSP mit Nachschub nach Erreichen eines Ziels**

* Betrachten nun MTSP mit Verfolgern, die nach Ende der Tour zum Ursprung zurückkehren muss

**Paper: de Freitas**

* Moving Target Traveling Salesman Problem (MT-TSP)
* Time Dependant Traveling Salesman Problem (TD-TSP).
* Anwendung: Systeme zur Überwachung und Steuerung der Menschenmenge (UAVs)
* Genetische Algorithmen erzielen die beste Leistung bei der Suche nach akzeptablen Lösungen für das Problem in Situationen mit eingeschränkter Zeit und Rechenleistung
* Dauer von einem Knoten zum anderen hängt davon ab, ab welchem Zeitpunkt die Tour gestartet ist oder von der Position der Knoten auf der Tour
* Spezieller Fall: Knoten bewegen sich mit bestimmter Geschwindigkeit abhängig von der Zeit
* “The approximation complexity of MT-TSP was studied by Hammar & Nilsson (1999), where it was shown that it cannot be approximated better than by a factor of two by a polynomial time algorithm unless P = NP, even if there are only two moving points in the instance.”
* Chouby (2013) zeigte, dass der von denen entwickelte genetische Algorithmus effizienter ist, als ähnlich modifizierte Greedy-Strategien
* Abeledo et al. (2013) schlägt polyedrischen Branch-Cut-and-Price-Algorithmus (BCP) vor, der besagt, dass die Polyedertheorie eine wichtige Rolle bei der Verbesserung von Algorithmen spielen bei MT-TSP und TD-TSP kann
* Englot et al. (2013) verglich die Lin-Kernighan Heuristik (LKH) mit einer Greedy-Heuriustik
  + Zeigte, dass die Leistungen von nicht-Greedy-Strategien von dem Geschwindigkeits-Verhältnis von den Zielen und dem Verfolger abhängen
  + LKH ist überlegen, wenn die Ziele sich mit niedriger Geschwindigkeit bewegen

**Heuristic Methods to solve TSP**

* **Genetische Algorithmen** (GA) zur Lösung des Problems
  + entwickeln von Kandidaten-Lösungen, bis eine gut genug zur Lösung des Problems ist
* Algorithmus:
  + Initiale Population entwickeln von Kandidaten-Lösungen
  + Wahl der Parents
  + Fitness der Parents mit Fitness-Funktion bestimmen
  + Rekombination, Crossover & Mutation
  + Vergleich Fitness der neuen Generation mit der alten
  + Survival of the Fitness-Prinzip
  + Stopp Kriterium bei einer gewissen Fitness oder nach einer maximalen Anzahl an Iterationen/Generationen (siehe Figure 2: GA Fluxogram)
* Schlechte Generationen können nach einigen Iterationen sehr gute Ergebnisse erzielen, ggf. sogar deutlich besser als lokale Optima
* 1. Problem: Repräsentation des MTTPS Problems als GA und dass die Repräsentation das Crossover, Selektion und Mutation nicht beeinflusst
* 2. & 3. Problem: Wahl der Operatoren Crossover & Mutation:
  + Für Variationen der Generationen verantwortlich
  + Beeinflusst die Konvergenz des Algorithmus
* 4. Problem: Selektion: Mit welcher Strategie werden die Parents in jeder Iteration ausgewählt?
* **Simulated Annealing** (SA) Heuristik, Prozess zum Finden von nahezu optimalen Lösungen, basiert auf der Idee des physikalischer Glühprozesses (Erhalt von niedrig-energetischen Zuständen von Metallen), Konzept:
  + Start bei einem hoch energetischen Zustand
  + Während der Ausführung werden fortlaufend weniger „schlechte“ Kandidatenlösungen akzeptiert, um sich mehr auf die „guten“ zu konzentrieren
* Stochastische Methode, um zu verhindern, dass lokale, nicht-global optimale Ergebnisse als optimal bewertet werden
  + Dafür werden sehr schlechte Lösungen verwendet, was zu optimalen Lösungen führen kann, wenn man diese lokalen Minima transponiert
* Algorithmus:
  + Generieren einer zufälligen Startpopulation
  + Lokale Suche um die initiale Lösung iterativ zu optimieren
  + Der Algorithmus startet mit einer initialen Temperatur T0, welche den initialen Energiezustand darstellt
  + In jeder Iteration werden *n* lokale Suchen auf die Lösungen angewendet, danach wird aktuelle Temperatur verringert
  + Sofern die lokale Suche eine bessere Lösung findet, als die aktuelle, wird diese übernommen
  + Ansonsten (Lösung x ist schlechter) wird auf eine Wahrscheinlichkeit in Abhängigkeit der Temperatur geprüft, um die Lösung zu übernehmen
    - Px=exp(-(f(i)-f(x))/Ti)
* Die Konvergenz des Algorithmus, um nahezu optimale Lösungen zu finden, wurde bereits bewiesen
* Durch die fallende Exponenten Gleichung (Temperatur sinkt in jeder Iteration) sinkt die Wahrscheinlichkeit, um schwache Lösungen zu akzeptieren
* Wichtig ist also der initiale Temperatur Parameter
  + Zu niedrig -> ggf. wird keine Lösung mehr akzeptiert
  + Zu hoch -> der Algorithmus braucht deutlich länger zum Finden einer optimalen Lösung, da sehr viele (unnötige) Lösungen akzeptiert werden
* Anpassungen deutlich einfacher, als bei GA
* Wichtig ist die Darstellung der Lösung und die Wahl des lokalen Suchalgorithmus
  + Unkomplizierte Darstellung der Lösung: Einfache Liste von Städten, sortiert in der Reihenfolge, in der die Tour durchgeführt wird
  + Wahl des lokalen Suchalgorithmus: Gestaltet sich als relativ schwierig. *K*-opt-Verfahren (2-opt, 3-opt) gehören zu den gängigsten, sie sind einfach und effektiv und beeinträchtigen nicht die Gesamtleistung. Es gibt allerdings auch GA in komplexeren Ansätzen
* **Ant Colony Optimization** (ACO) ebenfalls ein biologisch inspirierter Ansatz, um Optimierungsprobleme zu lösen
  + Bei Gruppeninsekten, in diesem Fall Ameisen, inspiriert, die gemeinsam kooperieren, um ihre Zielsetzung in der Natur erreichen zu können
  + Zur Kommunikation hinterlassen Ameisen eine chemische Spur (Botenstoffe)
* ACO nutzt dies, indem ebenfalls bei der Suche nach der optimalen Lösung der Weg zum Finden besserer Lösungen markiert wird
* Dabei generiert eine bestimmte Anzahl an Ameisen möglichen Lösungen und sie kommunizieren dann mit den Ablagerungen der Botenstoffe über die Qualität der Lösungen
* In der nächsten Iteration haben die Ameisen Richtlinien, um die Wanderwege mit den Botenstoffen auszuwerten, um dann damit eine bessere Generation zu generieren
  + Damit wird der Suchraum iterativ verfeinert, bis ein (fast) optimale Lösung gefunden wird
* Die Gebräuchlichsten Algorithmen sind Ant System (AS) (Dorigo et al., 1996) und Max-Min AS (Stützle & Hoos, 2000)
* Anwendung von TSP:
  + Verteilung der Ameisen in den Knoten
  + Lassen Ameisen wandern, sie versuchen dabei die Tour zu vervollständigen
  + Am Ende werden diese Wege zwischen den Knoten dann mit Botenstoffen versehen
  + Je kürzer die Tour, desto stärker der Botenstoff
  + In der nächsten Erkundung geht die Ameise den selben Weg nur zu einer bestimmten Wahrscheinlichkeit – proportional zur Menge an Botenstoffen auf der Kante
  + Jede Kante hat dabei eine gewisse Wahrscheinlichkeit, dass diese Kante benutzt wird
    - Verhindert lokale Minima, nach mehreren Iterationen werden bessere Lösungen gefördert
  + Andere Algorithmen führen nach jeder Iteration eine lokale Suche durch und optimieren damit die Pfade

Vergleich eigene Alg. mit moraes etc

TODO:

* Algorithmen
  + Hauptprobleme definieren (präzise)
  + 1D MTTSP – Problem ist bekannt (easer), Optimierungsproblem, bekommen Instanz X, Ziel: Route von Verfolger, Output (Reihenfolge), Varianten (Zeit, Strecke), Laufzeit verbessern?
  + 2D Fall – genauso wie 1D formulieren, was ist bekannt?, NP-hart, wenn statisch weiter NP-hart, da dann TSP, Approx schwere Aufgabe (wie einfach ist das Problem zu approximieren? Was haben andere darüber geschrieben), weiter np-vollständig wenn nur hor und vert bewegbar?
  + Formale Präsentation der Probleme, (Modellierung)