John Beighle

Department of Electrical and Computer Engineering
CSCE686 Advanced Algorithm Design
Spring Quarter 2020

Complex Optimization Algorithm Design Project

a) Problem Selection:

This CSCE686 final project proposal supports broader picture research/thesis domain-of-study objectives and will assist providing potential prospective inputs into that extended area of work. Thus, it is why this storyline was selected in hopes it helps expand, at least, creative possibilities and discussion points for the overarching goals of those efforts.

Thesis objectives, at this point, involve building a space-based game of sorts that involves contested space and pursuer evader satellites in low earth orbit. To fall in line with both the requirements of CSCE686 final project and thesis work the CSCE686 project will be proposed as the following scenario:

There is a Red (R) team recon Satellite that likes to fly in low orbit to take high resolution photographs of varying and multiple earth-based targets during each day. There are ten circular orbits, each within a 50km range of each other, that it can fly to according to the daily photo objectives and other associated risk factors. There are also two Blue (B) satellites that would like to capture the Red satellite, dock it, and reprogram its firmware to send back adjusted recon data. They are constantly trying to do this. R knows it is being chased but it must both continue its mission requirements and switch orbits on a set schedule, multiple time per day. So, every day it must build a flight schedule that drops it into the bottom 3 recon orbits for the daily scheduled durations, and then fly into higher orbits that use less fuel after each photo shoot. These shoots take at least 1 hour each. R also, due to risk factors, must also change orbits frequently when he is in a higher orbit and not engaged with taking pictures. These changes can be scheduled for the entire day as if B's made moves to capture R. However, Bs move multiple times per day making attempts to capture R. A sample of travel cost and constraints could look something like the following:

B Position			+4	X	+4	+4	X	+4		
Cost	+3	+2	+1	0	+1	+2	+3	+4	+5	+6
Orbits	1	2	3	4	5	6	7	8	9	10

24 Hour Photo Example Daily Mission Requirement

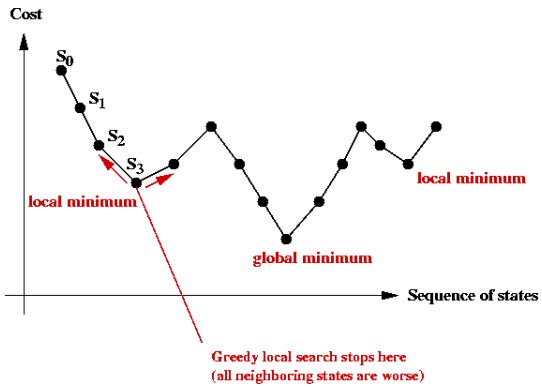
@ 0100 and @0300	@0500 and @0700	@0900 and @1200

R's 24-hour route plan can be created with a simple TSP algorithm. However, real life is not that easy because the risks factors change constantly with Bs continuous movement. So, a TASKORD project has been funded to create a dynamic TSP algorithm that can recalculate R's best route at any time needed during the day, anytime the risk factors change, i.e. the Bs have changed orbits and increased the capture risk profile for R. Space HQ wants this designed and documented before 3 July, 2020.

Note: The basic premise of the objective has evolved since original proposal due the fact that the problem space defined above is somewhat trivial. So, the problem is reformulated from its dynamic elements to a multi-objective TSP focused on optimization – which is still applicable to general thesis-based application as described above.

The multi-objective TSP problem looks to gain perspective from a general heuristic method used in optimization techniques; a good solution is more likely to be found nearby other good solutions than it is to

be found nearby an arbitrary solution. Qualifying "nearby" may be an approximation in terms of variation operators used by the search algorithm. [3] The simplest way to demonstrate this is with a hill-climber; however, the success of this approach is based on the strong assumption that within the solution space there exists a path through neighboring points to a global optimum that is monotonically increasing in value.[4] However, this is most often not the case. *What does the search landscape look like for a common TSP problem?* Often it is ***non-monotonic*** as in the below image.



Overcoming the non-monotonic nature of the landscape can addressed in a wide variety of ways. A common way is to probabilistically accept inferior solutions, by introducing simple multi-point searchers or multi-restart searchers. However, our solution will take a different approach by comparing two solutions that are common with multi-objective optimization (MOO), to circumvent local optima issues, using Pareto optimization. Here a solution x is superior (Pareto dominates) another solution x' if and only if it is at least as good as x' in all measures and better in at least one measure. Applying a Pareto hillclimber decomposes the problem into multiple objectives to address the local optima issues versus decomposing the search space to remove them, as in simulated annealing.

b) Problem Domain Specification

An unconstrained single-objective optimization (SOO) problem can be expressed:

$$\text{maximize } f(x)$$
$$\text{subject to } x \in X$$

where x is a discrete solution vector, and X is a finite set of feasible solutions, and f(x) maps X into R. A multi-objective combinatorial optimization (MOCO) problem can be expressed:

$$\text{"maximize" } f(x) = (f_1(x),\ldots,f_K(x))$$
$$\text{subject to } \quad x \in X$$

where the vector objective function f(x) maps X into R^K, where $K \geq 2$ is the number of objectives. "Maximize" is quoted because typically there does not exist a single solution that is maximal on all objectives. Additionally, the Pareto optimal set, $X^* \subseteq X$, can be used to find a set of solutions with the property:

$$\forall x^* \in X^* \cdot \nexists\, x \in X \cdot \succ x^*$$

Where $x \succ x^* \Leftrightarrow ((\forall i \in 1..K \cdot (f_i(x) \geq f_i(x^*)) \wedge (\exists i \in 1..K \cdot f_i(x) > f_i(x^*)))$. Where $x \succ x^*$ reads: x dominates x*, and solutions in the Pareto optimal set are also admissible. Additionally, for two solutions x and x', $x \sim x'$ if

and only if $\exists i \in 1..K \cdot f_i(x) > f_i(x') \wedge \exists j \in 1..K \cdot j \neq i \cdot f_j(x') > f_j(x)$. This pair is said to be incomparable and with respect to each other are nondominated. Since pairs like these are no worse than any other, if the selection operator allows, a hillclimber is free to move to it. This is the key of the Pareto hill-climber reducing the problem of local optima.

c) Algorithm Domain Selection & Specification

A simple neighborhood searching algorithm can assist in proving that multi-objectivizing can reduce local optima to a degree that can improve local searching. Since this work is focused on solutions for TSP using a hill-climbing technique will provide a nice contrasting capability between single and multi-objective approaches. The multi-objective hillclimber is like the Pareto Archived Evolution Strategy (PAES). The PAES is like a simple hillclimber in that it is a single-point hill-climber. Both simple hill-climbing and PAES accepts a neighbor of the current solution if it is not worse than any solution found so far. However, for PAES, worse means dominated and not merely higher cost.

As this research continues a mutation-only, genetic algorithm with deterministic crowding (DCGA) [5] will be introduced. This is a multi-point hill-climber that does not use recombination. The performance characteristics of each can be compared to the simple hillclimber and other baseline research results. The comparisons will help deliver the effects of minimizing local optimas and whether these effects provide performance benefits. Simulated Annealing would also be helpful to contrast because it incrementally adjusts the strictness with which it rejects worse solutions and works very well. It would serve as a useful comparison to multi-objectivization.

Example of data input file format from TSPLIB:

```
NAME:
kroB100
        TYPE: TSPLIB
        COMMENT: 100-city problem B (Krolak/Felts/Nelson)
        DIMENSION: 100
        EDGE_WEIGHT_TYPE : EUC_2D
        NODE_COORD_SECTION
        1 3140 1401
        2 556 1056
        3 3675 1522
```

Coordinates are specified for each node. Let x_i, y_i, be the coordinates of node i. The distance between two points i and j is computed as:

$$X_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Input D_i = set of cities C = {c_1, c_2,…, c_N} and for each pair {c_i, c_j} of distinct cities there is a distance d(c_i, c_j). The goal is to find an ordering π of the cities that minimizes the quantity.

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)}).$$

Output D_o = List of ordered nodes. A feasible/optimal solution with respect to the input domain.

John Beighle

d) Algorithm (program) Design and Refinement

Single-point hill-climber:

```
Initialization:   B ← ∅
                  x ∈ X ← Init(x)
                  B ← B ∪ x
Main Loop:        x' ∈ X ← Mutate(x)
                  if (Inferior(x', B) ≠ TRUE) {
                     x ← x'
                     B ← Reduce(x' ∪ B) }
Termination:      return Best(B)
```

Fig. 1. Generic pseudocode for hill-climbing algorithms

The Fig. 1. Hill-climbing pseudocode could be used for single or multi-objective search techniques.

x = current solution vector
B = minimal representation of the best solutions encountered so far
Mutate(x) returns a new solution x' made by variation of x

For a single-objective hillclimber: Inferior(), Reduce() and Best() are simple:

Inferior (x', B) returns true iff there is any element of the set B whose evaluation is greater than x'
Reduce(B) returns the set of equally maximum value elements of the set B
Best (B) returns any member of B because they are all equally good

For PAES multi-objective hill-climber the functions have Pareto versions as such:

Inferior(x', B) returns true iff there is any element of the set B that dominates x'
Reduce(B) returns the set of elements from B that are not dominated by another member of B
Best(B) returns the element of B that is maximal in the original single objective

Terminate: Terminate algorithms by constant *num_evals* evaluations.

TSP a popular academic optimization problem. To multi-objectivize this problem, sub-problems need be identified and solved. Normally, dependencies exist between cities of the problem which make it challenging to decompose. Using the concept of divide and conquer conveys a simple idea for decomposition, although not ideal, simply divide the problem into multiple sub-tours where each of these are to be minimized. This can be done several ways, and some are better than others depending on the data; however, applying a general method, sub-tours are broken into two-objective formulations to be minimized and defined by two cities. For input, set C = {c_1, c_2,..., c_N} of cities and for each pair {c_i, c_j} of distinct cities there is a distance d(c_i, c_j). Our goal is to find an ordering π of the cities that minimizes the quantity. This becomes:

$$\begin{aligned} \text{“minimize”}\ \mathbf{f}(\pi,a,b) &= (f_1(\pi,a,b), f_2(\pi,a,b)) \\ \text{where}\ f_1(\pi,a,b) &= \textstyle\sum_{i=\pi^{-1}(a)}^{\pi^{-1}(b)-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \\ \text{and}\ f_2(\pi,a,b) &= \textstyle\sum_{i=\pi^{-1}(b)}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + \\ &\quad \textstyle\sum_{i=1}^{\pi^{-1}(a)-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)}) \end{aligned}$$

Where a, b = two cities selected *apriori* and arbitrarily, and swapped if π(a) < π(b). The sum of the two objectives are the same as the quantity to be minimized in the single-point objective TSP equation:

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)}).$$

The optimum of the single-point objective TSP is coincident with at least one of the Pareto optima which is required as stated.

e) Implementation (NOT REQUIRED)

f) Design of experiments and Testing (NOT REQUIRED)

g) Extended Development

An NP-hard problem is “at least as hard as any NP-complete problem and quite possibly harder” [8]. Expanding upon this loose definition, an NP-hard problem cannot be solved while providing proof of the solution within polynomial time. An NP problem can have its proposed solution validated within polynomial time, i.e. O(n k). However, it is not possible for TSP to validate a solution within polynomial time because of the requirement to validate a tour is not just a Hamiltonian cycle but rather an optimum Hamiltonian cycle. This conditional requires all other feasible tours are known to be longer than the proposed solution. This, at its worst, requires a brute force check. Thus, TSP is classified as NP-hard under computational complexity theory [7].

TSP with single-point objective using a simple hillclimber can provide an upper bound on runtime. The time per iteration can be as low as O(n log n) noting that, if n is large, most swap calculations do not change from one iteration to the next. Thus, all distances are computed at the beginning, on each iteration, calculations are performed on only distances of pairs effected. If all calculation swaps are kept on a MaxHeap then there is the O(n) modifications in the MaxHeap cost. All told it takes O(n log n). The initialization before running the first iteration is $O(n^2)$.

Theorem 1:
The traveling salesman problem is NP-complete.

Proof:
First, we need prove that TSP belongs to NP. If we want to check a tour for credibility, we check that the tour contains each vertex once. Then we sum the total cost of the edges and finally check if the cost is minimum. This can be completed in p time. Thus, TSP belongs to NP.

Next, prove TSP is > NP-complete. Start by proving Hamiltonian cycle <= TSP. Given, Hamiltonian cycle is NP-Complete. Let G = (V, E) be an instance of Hamiltonian cycle. An instance of TSP is then constructed. We create the complete graph G’ = (V, E’) where E’ = {(i, j):i, j ∈ V and i ≠ j}. Thus, the cost function is defined as:

$$t(i,j) = \begin{cases} 0 \text{ if } (i, j) \in E, \\ 1 \text{ if } (i, j) \notin E. \end{cases}$$

Suppose a Hamiltonian cycle h exists in G. The cost of each edge in h is 0 in G’ as each edge belongs to E. Therefore, h has a cost of 0 in G’. Thus, if graph G has a Hamiltonian cycle then graph G’ has a tour of 0 cost. Conversely, assume G’ has a tour h’ of cost at most 0. The cost of edges in E’ are 0 and 1. So, each edge must have a cost of 0 as the cost of h’ is 0. Concluding that h’ contains only edges in E. It is proven G has a Hamiltonian cycle if and only if G’ has a tour of cost at most 0. Thus, TSP is NP-complete.

John Beighle

h) Report - TBD

REFERENCES:

[1] Generating Human-readable Algorithms for the Travelling Salesman Problem, P. Ryser-Welch, J. Miller, S. Asta 2015
[2] International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization) Vol. 4, Issue 1, January 2016
[3] T. Jones. Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, University of New Mexico, Albuquerque, 1995.
[4] R. Bellman. Dynamic programming and multi-stage decision processes of stochastic type. In Proceedings of the second symposium in linear programming, volume 2, pages 229{250, Washington D.C., 1955. NBS and USAF.
[5] 11. S. W. Mahfoud. Niching methods for genetic algorithms. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1995. IlliGAL Report 95001.
[6] Reducing Local Optima in Single-Ob jective Problems by Multi-ob jectivization J. Knowles, R. Watson, D. Corne
[7] Grefenstette, J., Gopal, R., Rosmaita, B., and Van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. In Proceedings of the first International Conference on Genetic Algorithms and their Applications, pages 160–168.
[8] Hillar, C. J. and Lim, L.-H. (2013). Most tensor problems are np-hard. Journal of the ACM (JACM), 60(6):45.
[9] The Traveling Salesman Problem, Chpt 10