Employing Supportive Coevolution for the Automated Design and Configuration of Evolutionary Algorithm Operators and Parameters

Nathaniel Kamrath

Evolutionary Computing – Fall 2021

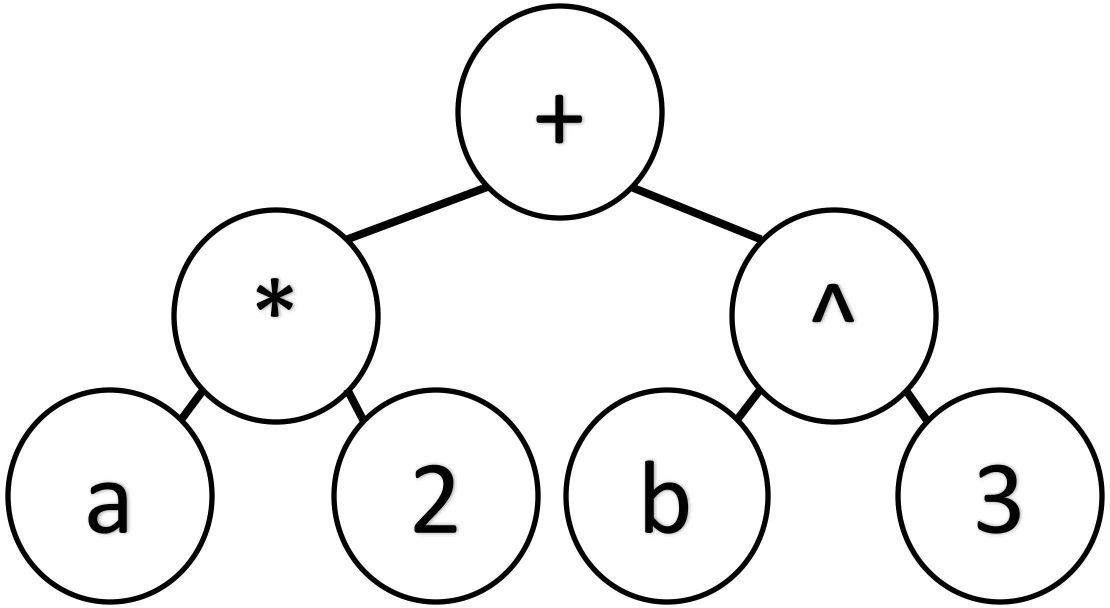
AUBURN UNIVERSITY

Outline

- Introduction
- Using SuCo to Evolve Self-Configuring Crossover
- The Automated Design of Local Optimizers for Memetic Algorithms Employing SuCo
- The Evolution of Deme Specific Local Optimizers in a Diffusion Memetic Algorithm Employing SuCo
- Solving the Traveling Thief Problem with a Diffusion Memetic Algorithm Employing SuCo

Introduction GP

Tree GP:
(a*2) + (b^3)

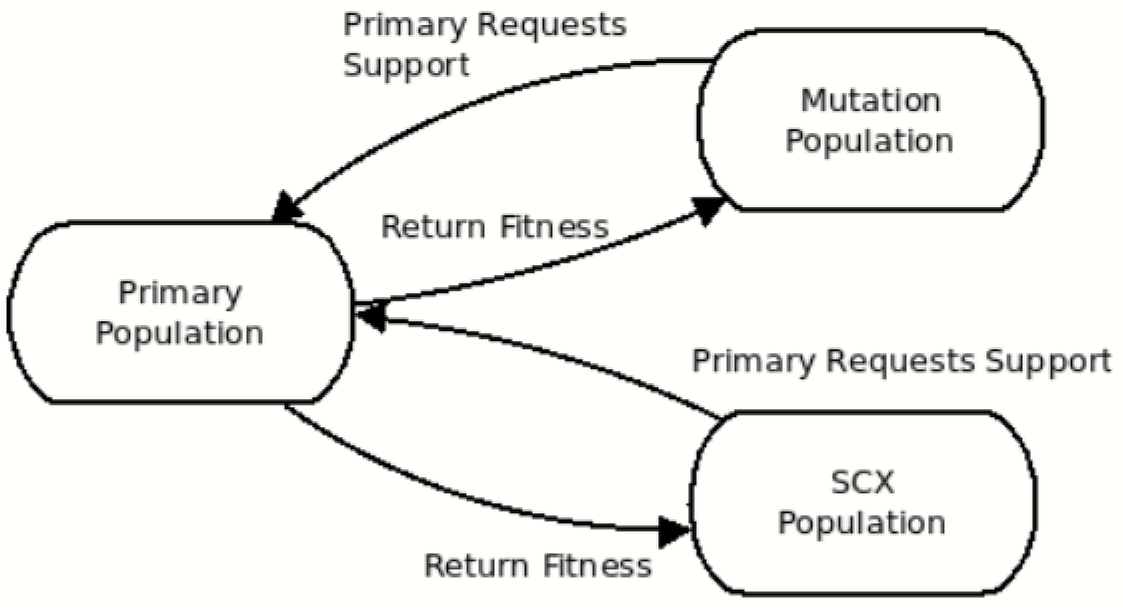


Stack GP:
(a*2) + (b^3)

EXEC	INPUT	INT
	a, b	
INPUT.TO_INT	b	a
INT.2	b	2, a
INT.MULTIPLY	b	2a
INT.3		b, 2a
INPUT.TO_INT		b, 3, 2a
INT.POWER		b^3, 2a
INT.ADD		(b^3 + 2a)

Introduction SuCo

- Supportive Coevolution (SuCo)
  - Technique for online parameter and operator evolution

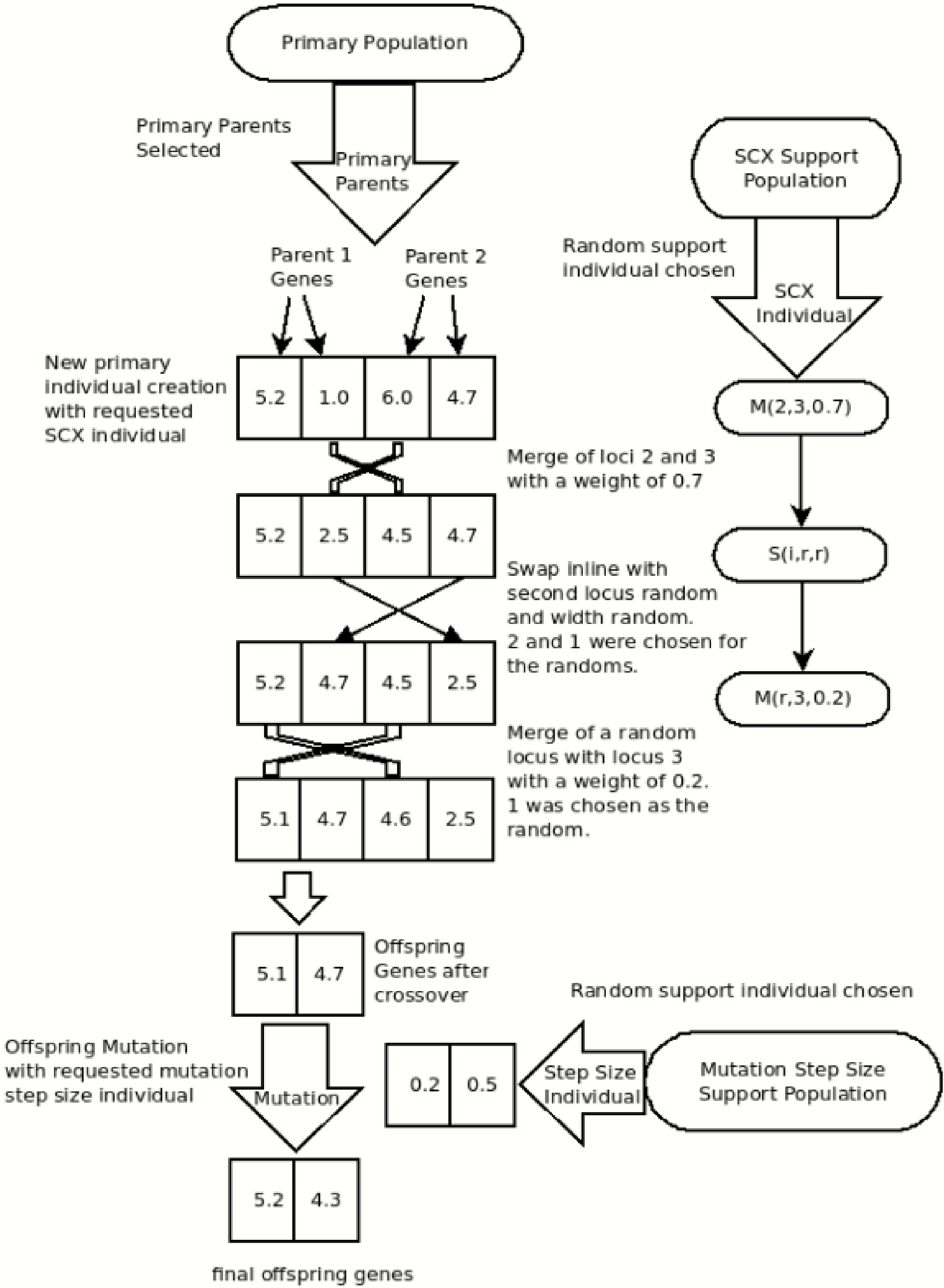


Using SuCo to Evolve SCX

- Self-Configuring Crossover (SCX)
  - Self-adaptive technique for dynamic crossover operator design
  - Linear Genetic Programming (LGP) structure
  - Primitives
    - Swap – represents crossovers that move genetic information between parents and between positions in a single parent (n-point, uniform, permutation)
    - Merge – represents crossovers that create genetic material by combining genes (arithmetic crossover)

SCX Design

- SuCo + SCX
  - Two support populations
    - SCX for crossover operator
    - Mutation Step Size parameter



SuCo SCX Experiments

- Rastrigin $$An + \sum_{i=1}^{n} [x_i^2 - 10cos(2\pi x_i)] \; \forall x \; \in \; [-5.12, 5.12]$$
- Rosenbrock $$\sum_{i=1}^{n-1} [A(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \; \forall x \; \in \; [-5, 10]$$
- Shifted Rastrigin $$z = x + o \;\;, x = [x_1, x_2, ...x_n], o = [o_1, o_2, ...o_n]$$

SuCo SCX Results

- Man-Whitney U tests with α = 0.01
  - SuCo Mutation + SuCo SCX was found to be the best combination on Rastrigin and Rosenbrock
  - SuCo Mutation + SuCo SCX was the same as SuCo Mutation + SA SCX on Shifted Rastrigin

	Static Mutation	SuCo Mutation
Arithmetic	-55.925 (6.4379)	-61.249 (1.5353)
SA-SCX	-0.0072 (0.00465)	-0.01711 (0.00690)
SuCo SCX	-0.00051 (0.00233)	-0.00025 (0.00078)

Rastrigin Results

	Static Mutation	SuCo Mutation
Arithmetic	-0.12568 (0.18911)	-0.15197 (0.19091)
SA-SCX	-0.02718 (0.01818)	-0.02366 (0.01194)
SuCo SCX	-0.30794 (0.46271)	-0.02579 (0.01022)

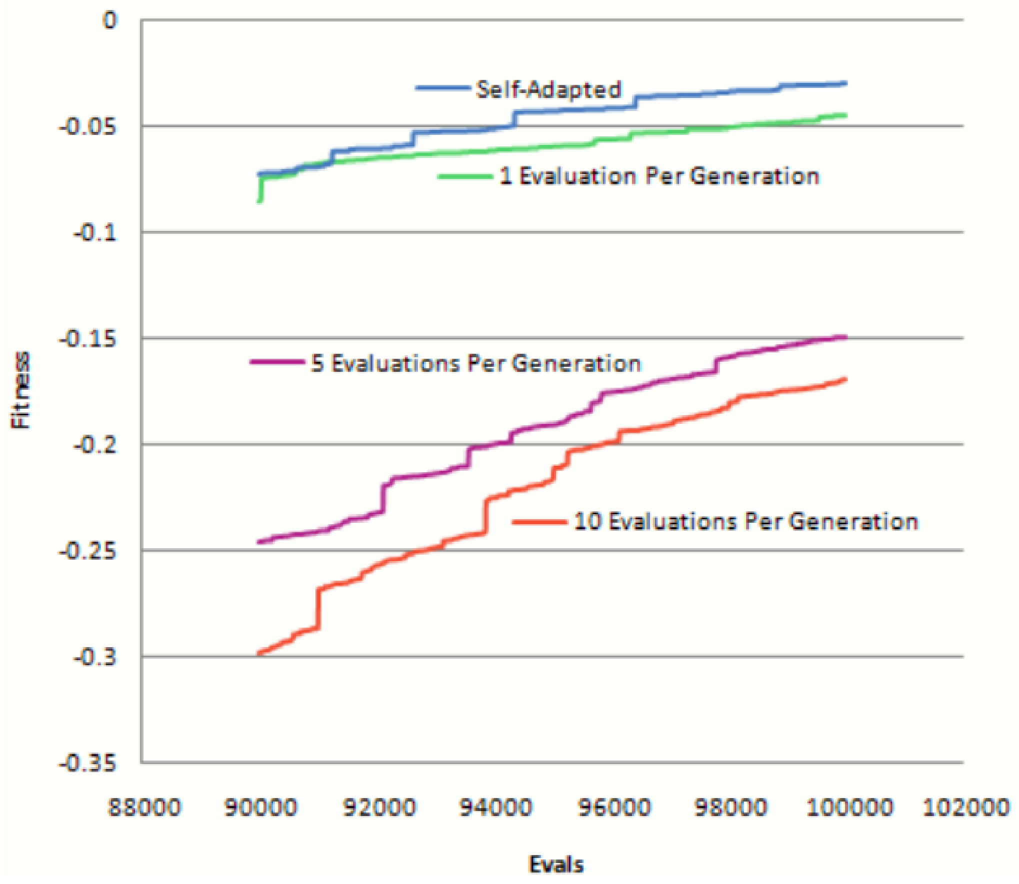
Shifted Rastrigin Results

	Static Mutation	SuCo Mutation
Arithmetic	-71.873 (39.489)	-85.468 (41.840)
SA-SCX	-28.999 (22.847)	-24.444 (23.446)
SuCo SCX	-22.638 (23.201)	-17.941 (22.824)

Rosenbrock Results

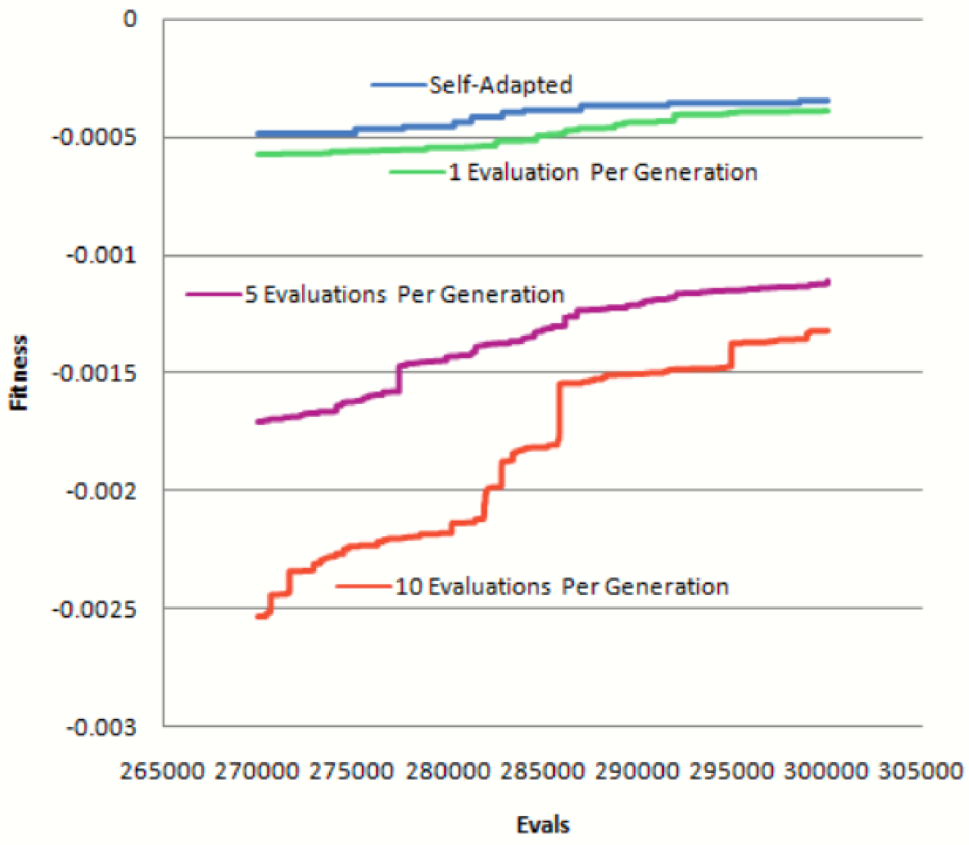
SuCo SCX Discussion

- Evals Per Generation
  - Mean best fitness over 50 runs on Shifted Rastrigin
  - Less evaluations per generation results in faster adjustments from support population



SuCo SCX Discussion

- 300k evaluations averaged over 10 runs
  - Relative positions unchanged
  - Self-adapted vs 1 Eval Per Generation



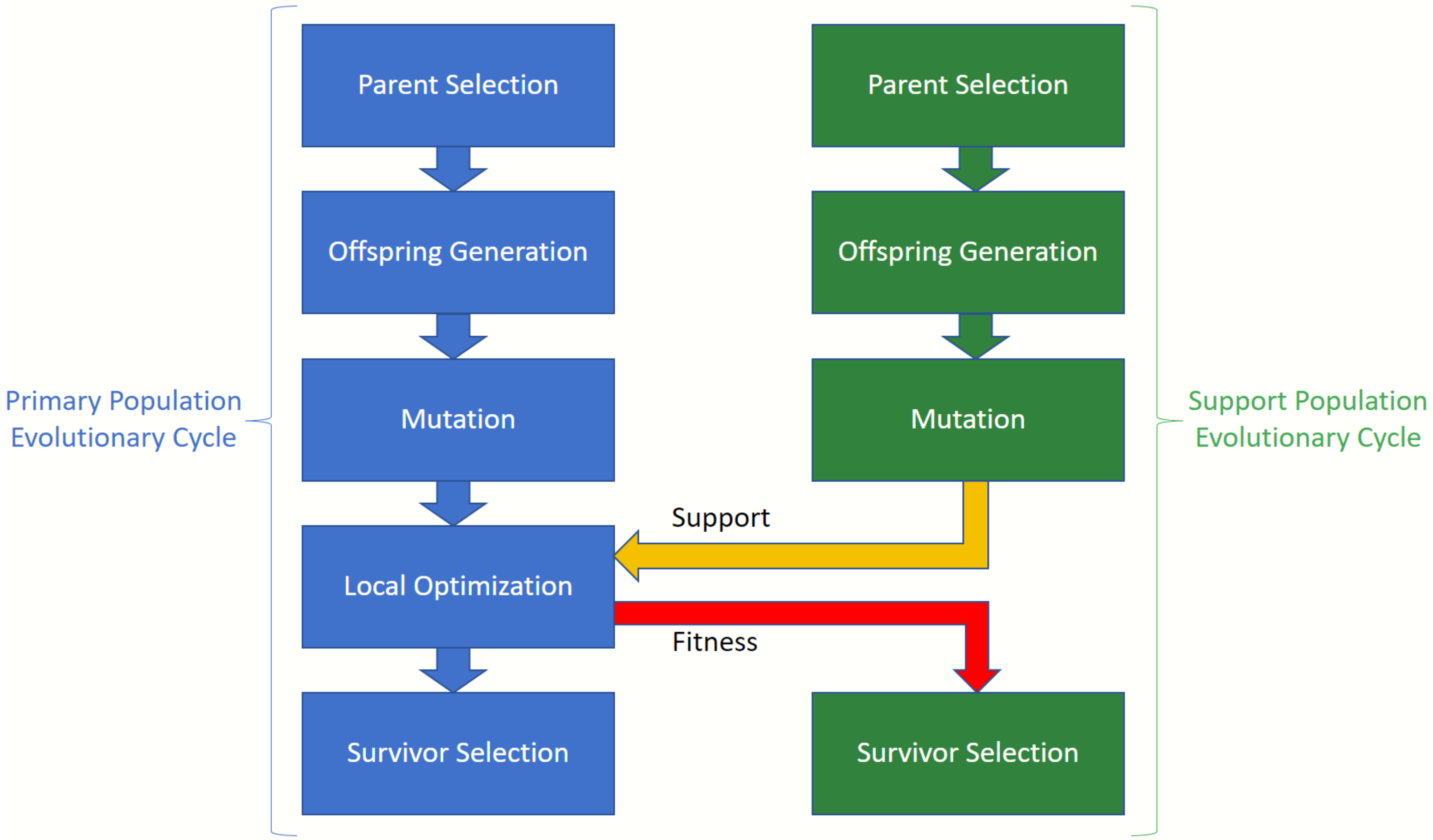
SuCo SCX Conclusions

- SuCo + SCX can improve EA performance through flexibility
  - Can outperform self-adaptation
- SuCo is a promising way of evolving multiple parameters and operators

SuCo MA

- Supportive Coevolution Memetic Algorithm (SuCo MA)
- Single support population of local optimizers encoded using PushGP to create a generative hyper-heuristic
  - PushGP is a stack based, linear programming language that is well suited for program evolution
  - Support population explores the space of local optimization algorithms

SuCo MA Structure



PushGP Vector Stack Instructions

AUBURN UNIVERSITY

Push Instruction	Inputs	Output	Operation
vector.add	vector, vector	vector	Vector addition
vector.subtract	vector, vector	vector	Vector subtraction
vector.multiply	vector, vector	vector	Vector multiplication
vector.scale	vector, float	vector	Scale a vector by a float value
vector.index_add	vector, int, float	vector	Add a value to vector member at index
vector.index_subtract	vector, int, float	vector	Subtract a value from vector member at index
vector.index_multiply	vector, int, float	vector	Multiply vector member at index by float value
vector.index_divide	vector, int, float	vector	Divide vector member at index by float value
vector.duplicate	vector	vector	Duplicates the top vector item
vector.random		vector	Creates a vector of random floats within a specified range
vector.random_unit		vector	Creates a random unit vector
vector.magnitude	vector	float	Computes the magnitude of a vector
vector.between	vector, vector	vector	Creates a vector between two vectors
vector.reverse	vector	vector	Reverses a vector
vector.length	vector	float	Gets the length of a vector
vector.constant		vector	Creates a constant vector
vector.from_float	float	vector	Creates a vector of where every element is the input value
vector.at_index	vector	float	Gets an element from a vector

SuCo MA Experiments

- Shift Vector $z = x + o \ \ , x = [x_1, x_2, ...x_n], o = [o_1, o_2, ...o_n]$
- Shifted Rastrigin $\sum_{i=1}^{n} [z_i^2 - 10cos(2\pi z_i) + 10] \ \forall z \ \in \ [-5.12, 5.12]$
- Shifted Rosenbrock $\sum_{i=1}^{n-1} [100(z_{i+1} - z_i^2)^2 + (1 - z_i)^2] \ \forall z \ \in \ [-5, 10]$

SuCo MA Results

Algorithm	50D	100D	200D
Traditional EA	**-0.071 (0.025)**	-0.48 (0.11)	-8.14 (1.14)
MA with Hill Climb	**-0.085 (0.022)**	-0.47 (0.12)	-8.18 (1.65)
MA with SuCo	**-0.075 (0.022)**	**-0.38 (0.080)**	**-7.05 (1.02)**

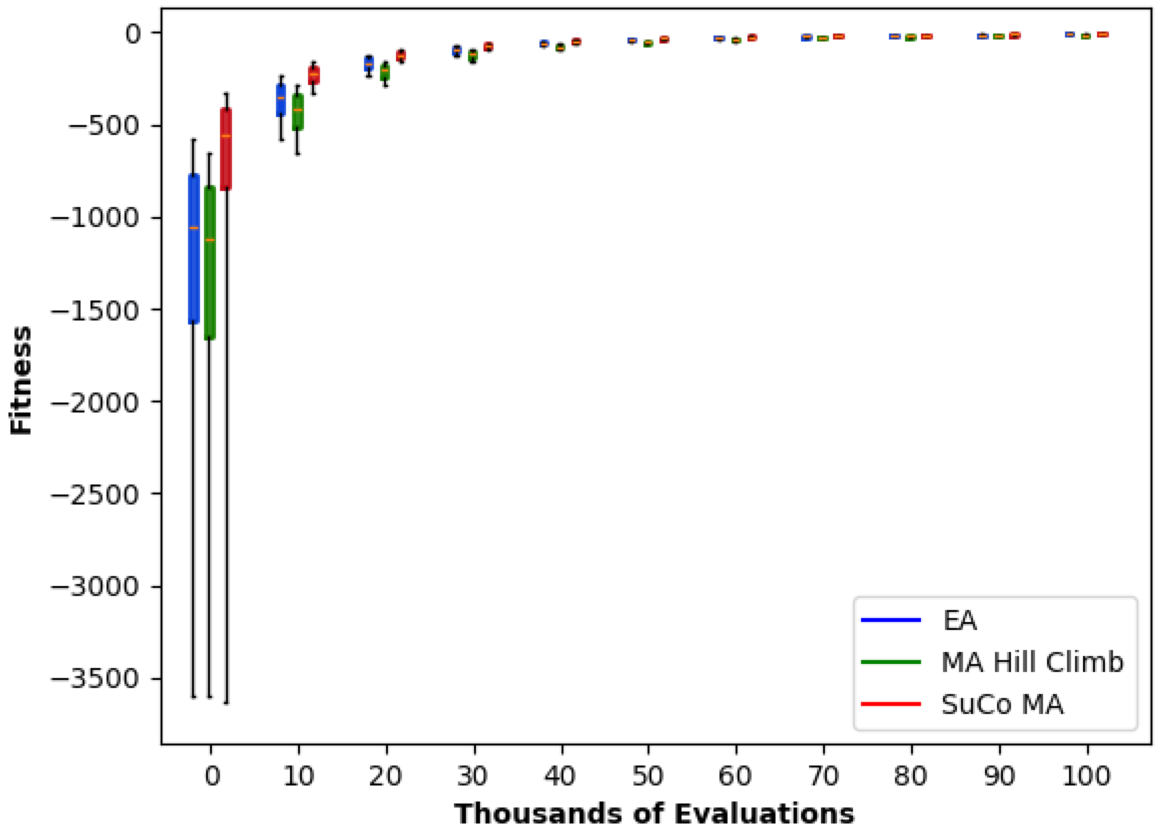
Shifted Rastrigin Results

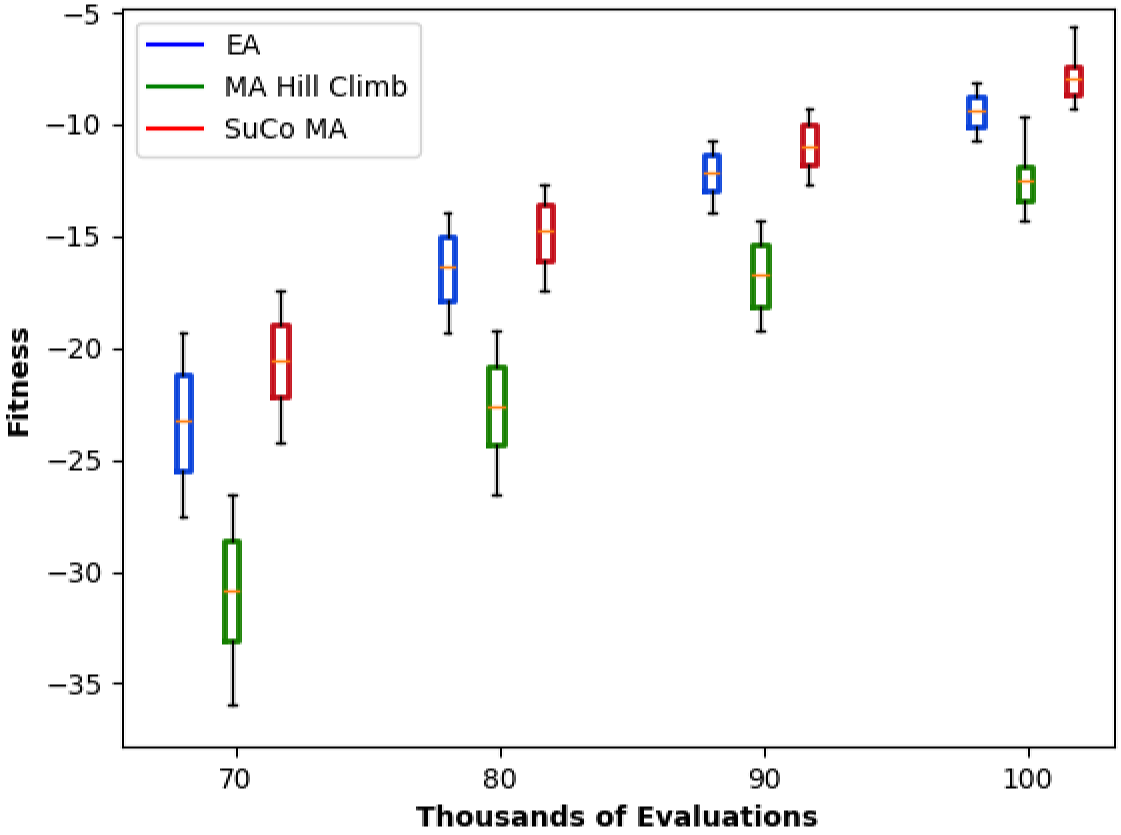
Algorithm	50D	100D	200D
Traditional EA	-75.57 (39.43)	-179.37 (66.48)	-389.26 (98.51)
MA with Hill Climb	-119.29 (71.77)	-292.77 (125.95)	-538.73 (142.05)
MA with SuCo	**-45.99 (13.68)**	**-123.02 (41.52)**	**-268.47 (56.50)**

Shifted Rosenbrock Results

SuCo MA Results

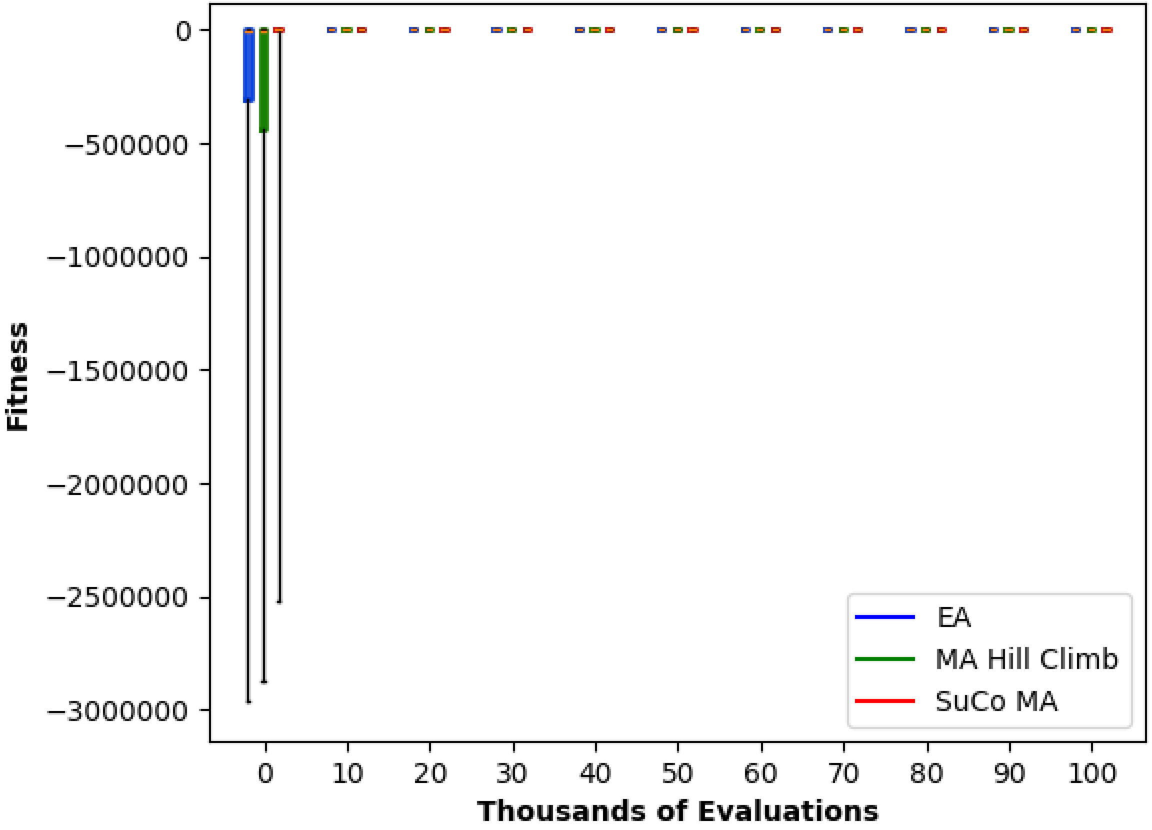
- Fitness vs. evals for Shifted Rastrigin in 200 dimensions

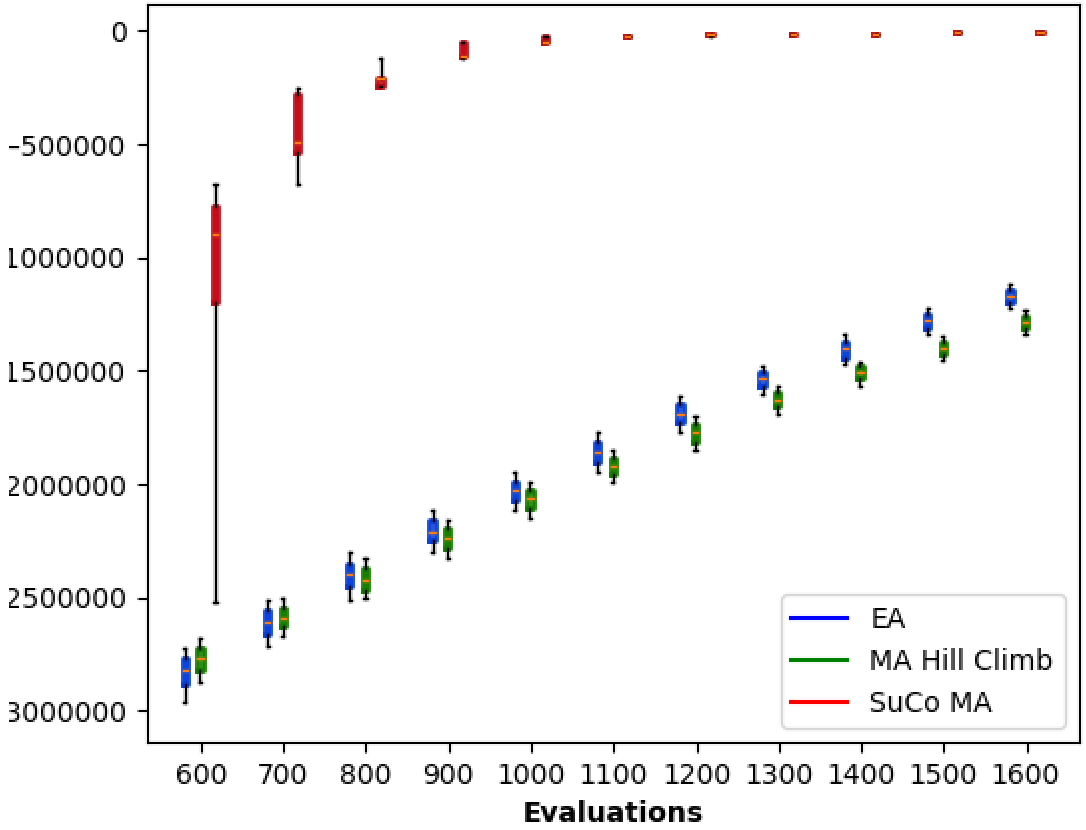




SuCo MA Results

- Fitness vs. evals Shifted Rosenbrock in 50 dimensions





SuCo MA Evolved Optimizers

Problem	Push Program
Rastrigin 200D	(vector.constant vector.between vector.at_index integer.duplicate float.add vector.random_unit vector.at_index vector.between vector.at_index)
Rosenbrock 50D	(float.random vector.at_index float.sin 26 vector.random vector.duplicate vector.divide vector.index_divide vector.multiply float.duplicate float.random)

SuCo MA Discussion

- Evaluations Per Local Optimizer
  - Each local optimizer can be applied to multiple primary individuals

	Shifted Rastrigin 200D	Shifted Rosenbrock 50D
1 Evaluation	**-7.05 (1.02)**	**-45.99 (13.68)**
2 Evaluations	-8.33 (1.84)	-50.42 (12.84)
3 Evaluations	-8.36 (1.35)	-52.18 (26.08)
4 Evaluations	-9.11 (1.29)	-53.54 (29.80)
5 Evaluations	-9.69 (1.91)	-54.11 (29.91)

SuCo MA Discussion

AUBURN UNIVERSITY

- Selective Optimization
  - It is not necessary to perform optimization every generation of the primary population

Generations	Shifted Rastrigin 200D	Shifted Rosenbrock 50D
0	-11.97 (1.84)	-55.18 (28.15)
4	-7.74 (1.32)	-54.59 (22.81)
8	-7.21 (1.10)	-54.76 (27.11)
12	**-7.05** (1.02)	-51.95 (20.76)
16	-7.08 (1.68)	**-45.99 (13.68)**
20	-7.72 (1.62)	-54.72 (24.17)
24	-7.73 (1.51)	-53.72 (22.46)
28	-7.60 (1.19)	-59.04 (31.38)

SuCo MA Discussion

- Handling Optimization Results
  - After local optimization, the resulting genes and fitness value can be handled in different ways

Problem	Fitness	Fitness and Genes	New Offspring
Shifted Rastrigin 200D	-2163.83 (46.16)	-7.44 (1.63)	**-7.05 (1.02)**
Shifted Rosenbrock 50D	-27619.38 (1051.33)	-49.95 (13.81)	**-45.99 (13.68)**

SuCo MA Conclusions

- Using SuCo to evolve PushGP encoded local optimizers can improve performance
- This can reduce MA configuration by automatically designing optimization algorithms
- It can also improve performance since the local optimization strategy can adapt to the current state of the EA throughout the evolutionary run

SuCo-Dif-MA

- Extends SuCo MA by adding a diffusion model to create SuCo-Dif-MA
- Diffusion model can encourage the evolution of deme specific strategies

SuCo-Dif-MA PushGP Vector Instructions

AUBURN UNIVERSITY

Instruction	Input	Output	Operation
vector.+	vector, vector	vector	adds two vectors
vector.-	vector, vector	vector	subtracts two vectors
vector.*	vector, vector	vector	multiplies two vectors
vector./	vector, vector	vector	divides two vectors
vector.SCALE	vector, float	vector	scales a vector by a double
vector.INDEX+	vector, int, float	vector	adds to vector value at index
vector.INDEX-	vector, int, float	vector	subtracts from vector value at index
vector.INDEX*	vector, int, float	vector	multiplies vector value at index
vector.INDEX/	vector, int, float	vector	divides vector value at index
vector.DUP	vector	vector	duplicates the top vector
vector.RAND	none	vector	creates a random vector
vector.RAND_UNIT	none	vector	creates a random unit vector
vector.MAGNITUDE	vector	float	calculates the magnitude of a vector
vector.BETWEEN	vector, vector	vector	calculates the midpoint vector
vector.FROM_FLOAT	float	vector	creates a vector from a float
vector.AT_INDEX	vector, int	float	gets value at index
vector.MIX	vector, vector	vector	mixes two vectors together
vector.SIN	vector	vector	takes the sin of a vector by index
vector.COS	vector	vector	takes the cos of a vector by index

SuCo-Dif-MA Experiments

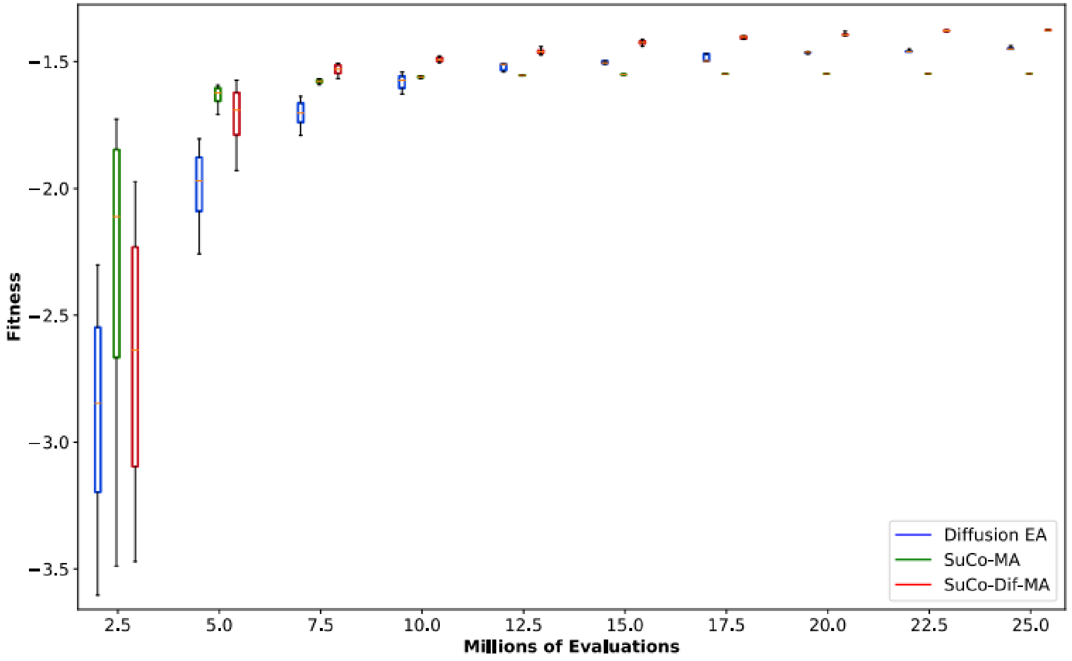
- Shift Vector $z = x + o \;\; , x = [x_1, x_2, \ldots x_n], o = [o_1, o_2, \ldots o_n]$
- Shifted Schwefel $-\frac{1}{100n}\sum_{i=1}^{n} z_i \sin(\sqrt{|z_i|}) + 4.189828872724339 \; \forall z \;\in\; [-500, 500]$
- Shifted Rosenbrock $\sum_{i=1}^{n-1} [100(z_{i+1} - z_i^2)^2 + (1 - z_i)^2] \; \forall z \;\in\; [-5, 10]$

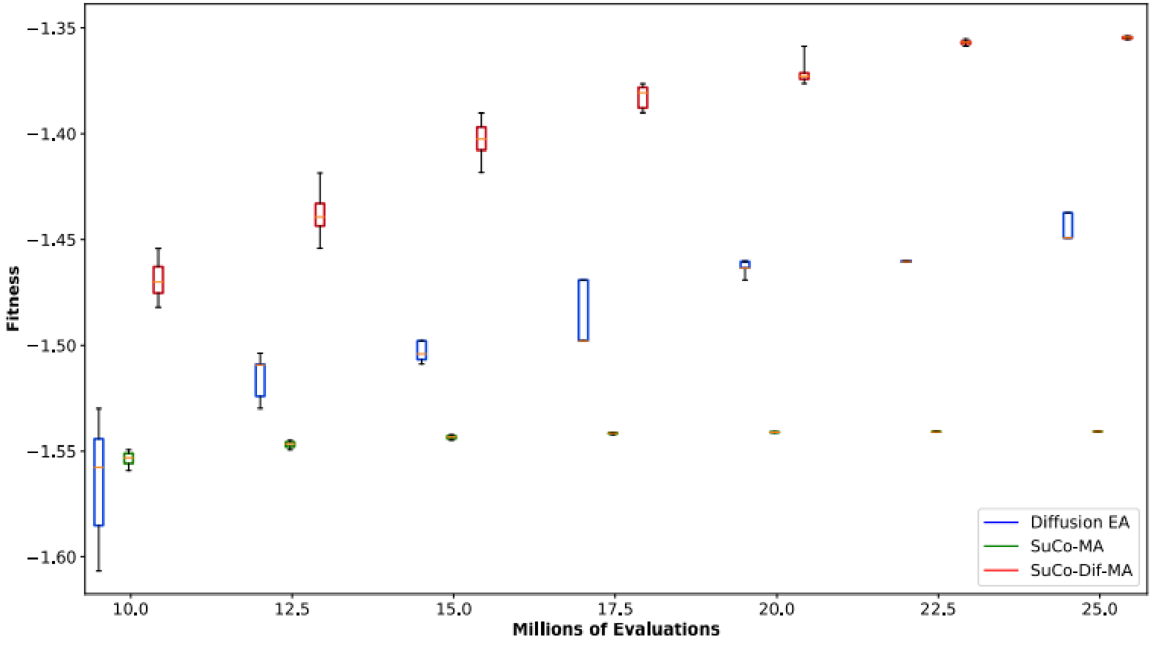
SuCo-Dif-MA Results

	Schwefel 50D	Schwefel 200D	Rosenbrock 50D	Rosenbrock 200D
Diffusion EA	**-0.87 (0.17)**	-1.48 (0.08)	-41.54 (5.57)	-241.60 (25.10)
SuCo-MA	-1.25 (0.16)	-1.55 (0.15)	-39.09 (17.26)	-262.32 (79.70)
SuCo-Dif-MA	**-0.83 (0.14)**	**-1.38 (0.07)**	**-10.33 (16.51)**	**-91.10 (67.26)**

SuCo-Dif-MA Results

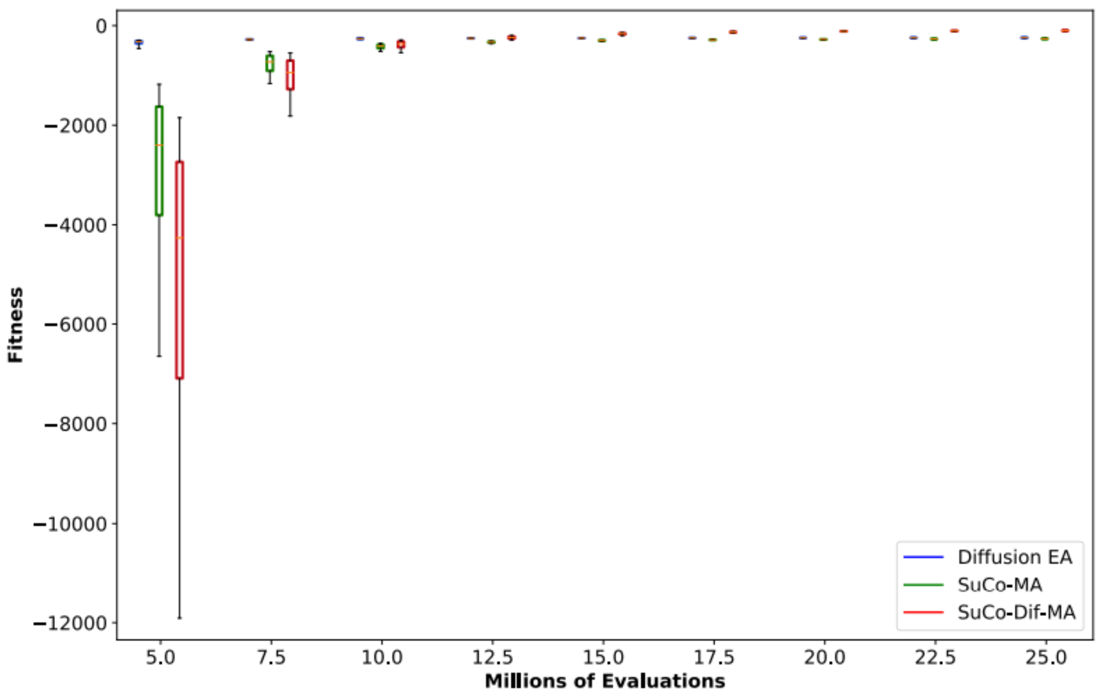
- Shifted Schwefel fitness vs evals in 200 dimensions

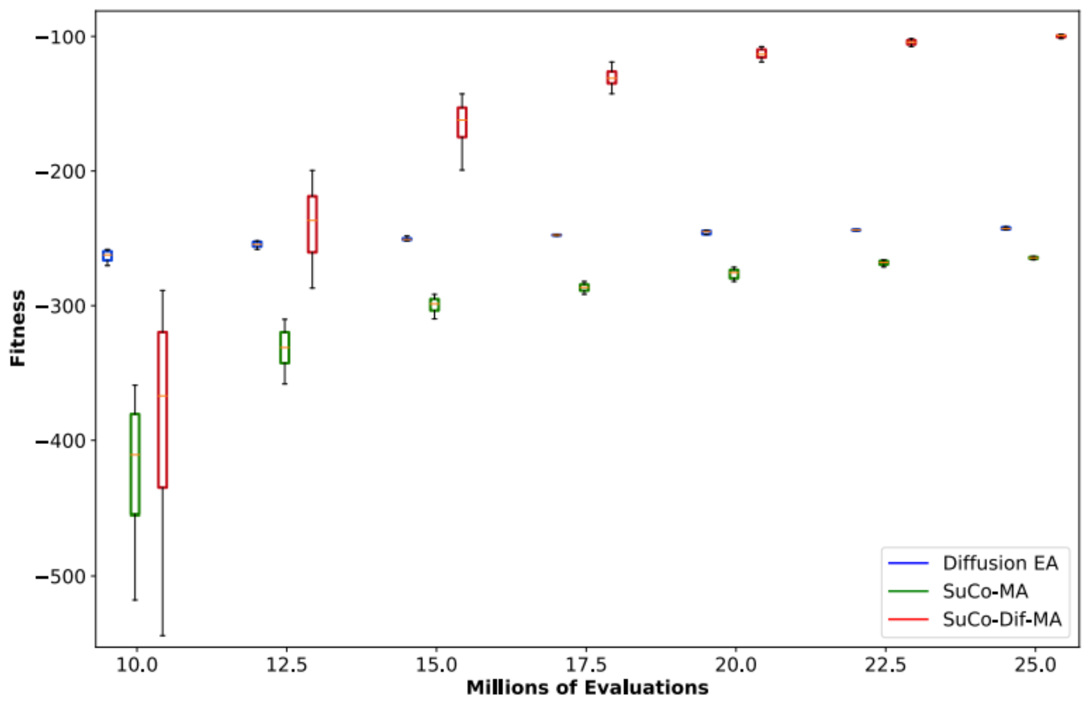




SuCo-Dif-MA Results

- Shifted Rosenbrock fitness vs evals in 200 Dimensions



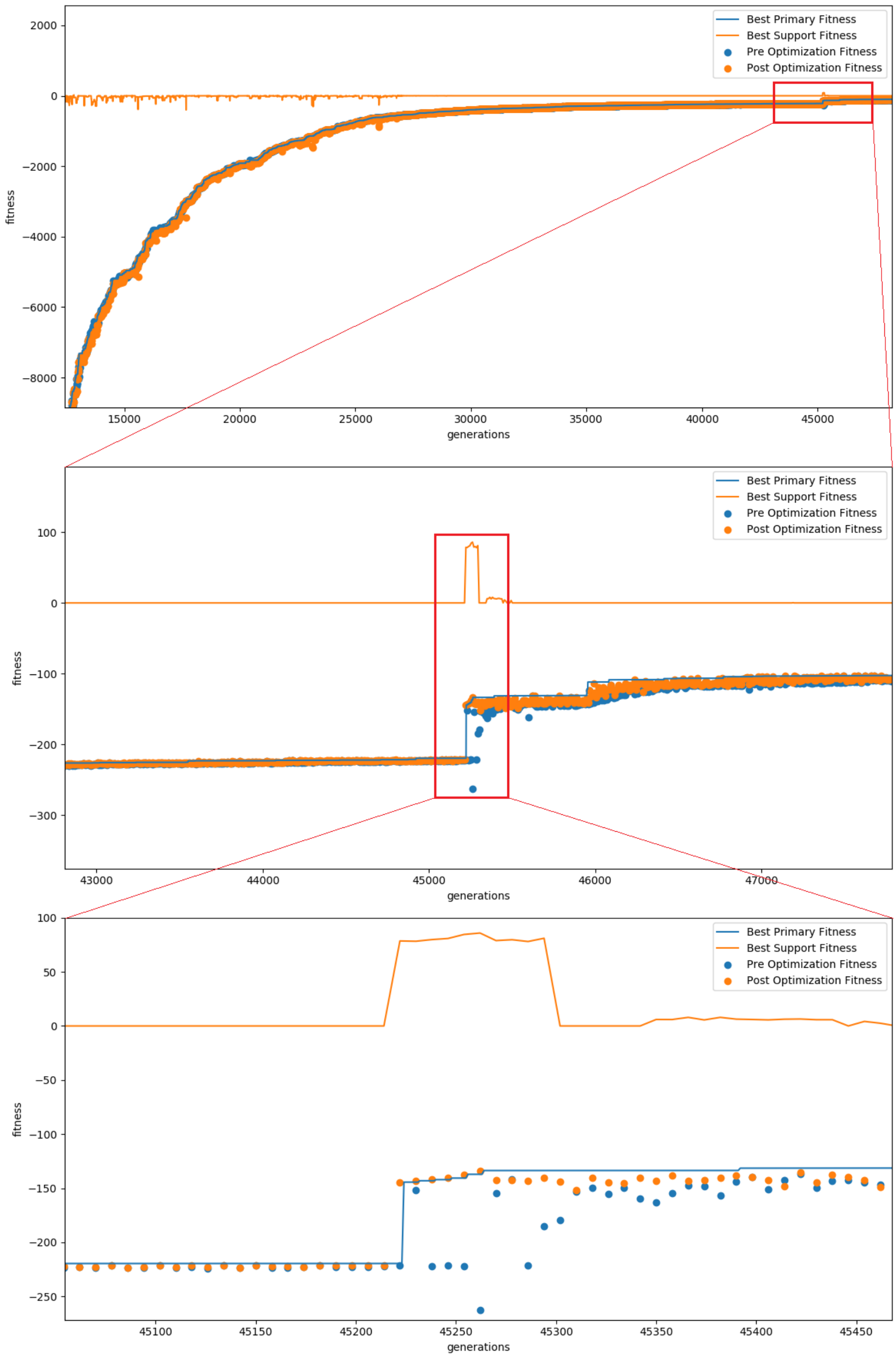


SuCo-Dif-MA Discussion

- Local optimizer fitness function
  - In contrast with SuCo-MA, SuCo-Dif-MA performance was improved by using the maximum improvement found across 4 optimization trials against different primary individuals
  - Encouraged deme specific strategies

High Performance Optimizer Discovery

- High performance optimizer discovery can take time, but can yield large improvements in fitness



SuCo-Dif-MA Discussion

- Optimization Frequency
  - Improvement on previous work's study of Selective Optimization parameter

Optimization Frequency	Rosenbrock 200D
0.2	-168.37 (55.66)
0.125	-94.64 (62.87)
0.1	**-91.09 (67.26)**
0.0833	-98.17 (76.66)
0.0667	-130.91 (85.04)

SuCo-Dif-MA Conclusions

- SuCo-Dif-MA can improve performance when compared to SuCo-MA
- High performance, deme specific optimization strategies can be evolved, but their evolution takes time and is not consistent

SuCo-Dif-MA and TTP

- Traveling Thief Problem (TTP) is a combination of the classical Traveling Salesman Problem (TSP) and the Knapsack Problem (KP)
- High quality TTP solutions must consider both the TSP and KP sub-problems simultaneously
- TTP has been shown to be a good benchmark for simulating the complexity and difficulty of real world problems.

TTP Problem Definition

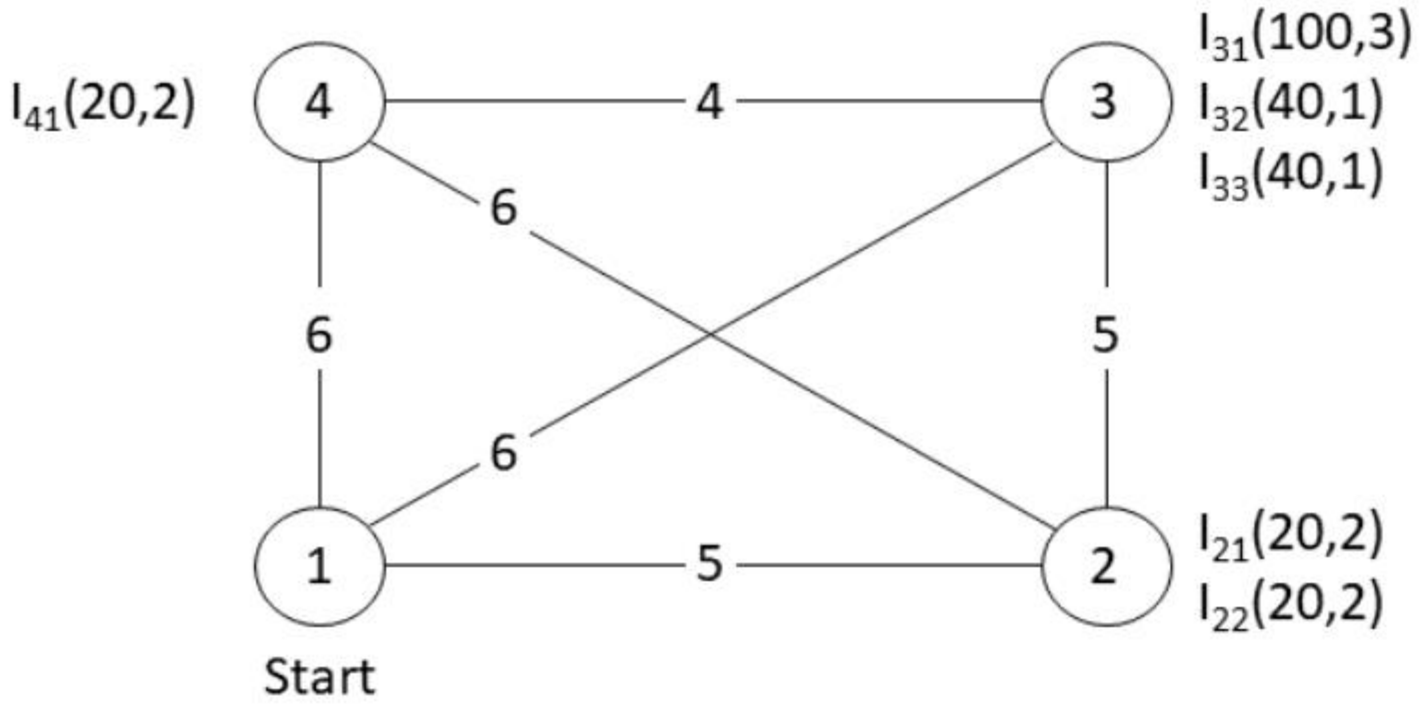
- An instance of TTP is defined by the following parameters
  - n: number of cities
  - m: number of items
    - Each item has profit p_k and weight w_k
  - Knapsack has capacity W
  - Renting rate R
  - Thief has a minimum and maximum velocity v_{min} and v_{max}
  - A valid TTP solution visits every city exactly one time while filling the knapsack without exceeding the capacity and then traveling back to the starting city.

TTP Problem Definition

- Thief velocity calculation: $$v_{x_i} = v_{max} - \frac{v_{max} - v_{min}}{W} \cdot w_{x_i}$$
- Total knapsack profit: $$g(z) = \sum_{k=1}^{m} p_k \cdot z_k$$
- Total thief travel time: $$f(x,z) = \frac{d_{x_n x_1}}{v_{x_n}} + \sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{v_{x_i}}$$
- Objective Function: $$G(x,z) = g(z) - R \cdot f(x,z)$$

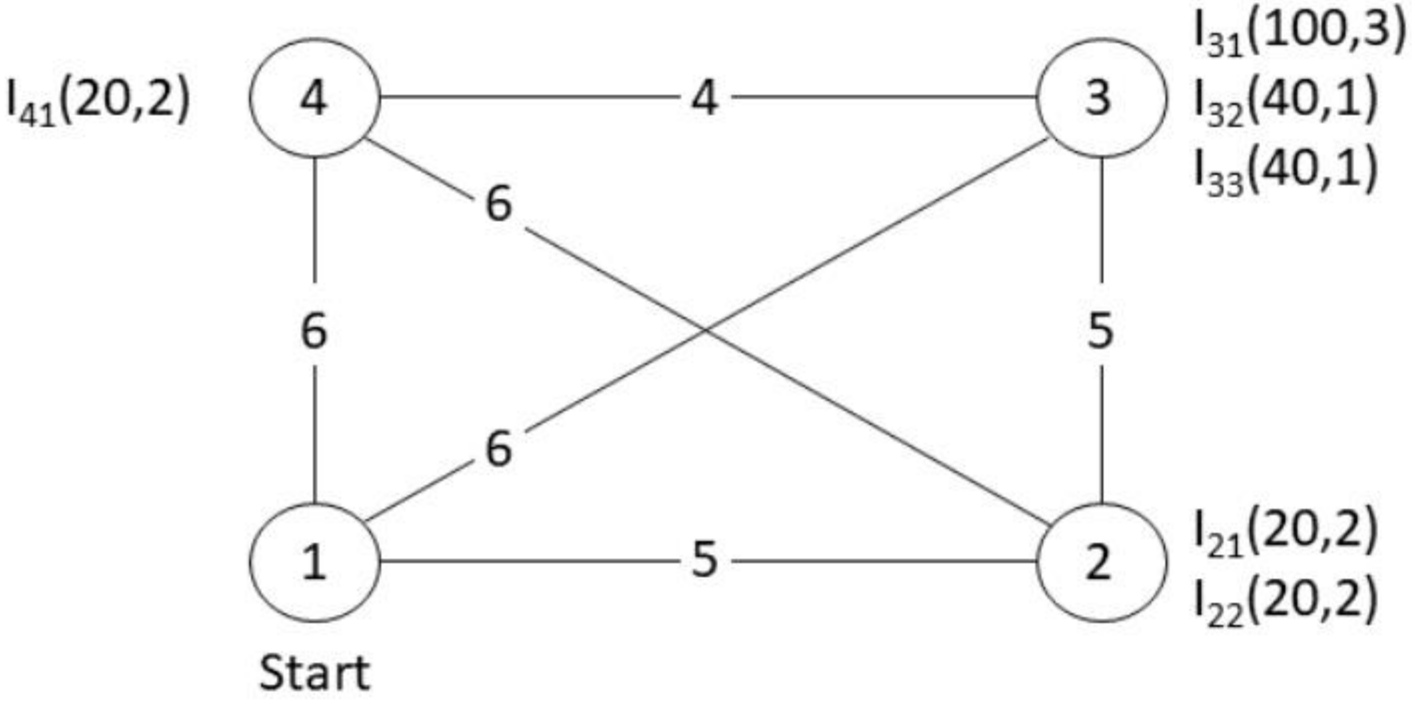
TTP Example

- $W = 3, v_{min} = 0.1, v_{max} = 1.0$
- Tour $x = (1, 2, 4, 3)$
- Packing plan $z = (0, 0, 0, 1, 1, 0)$



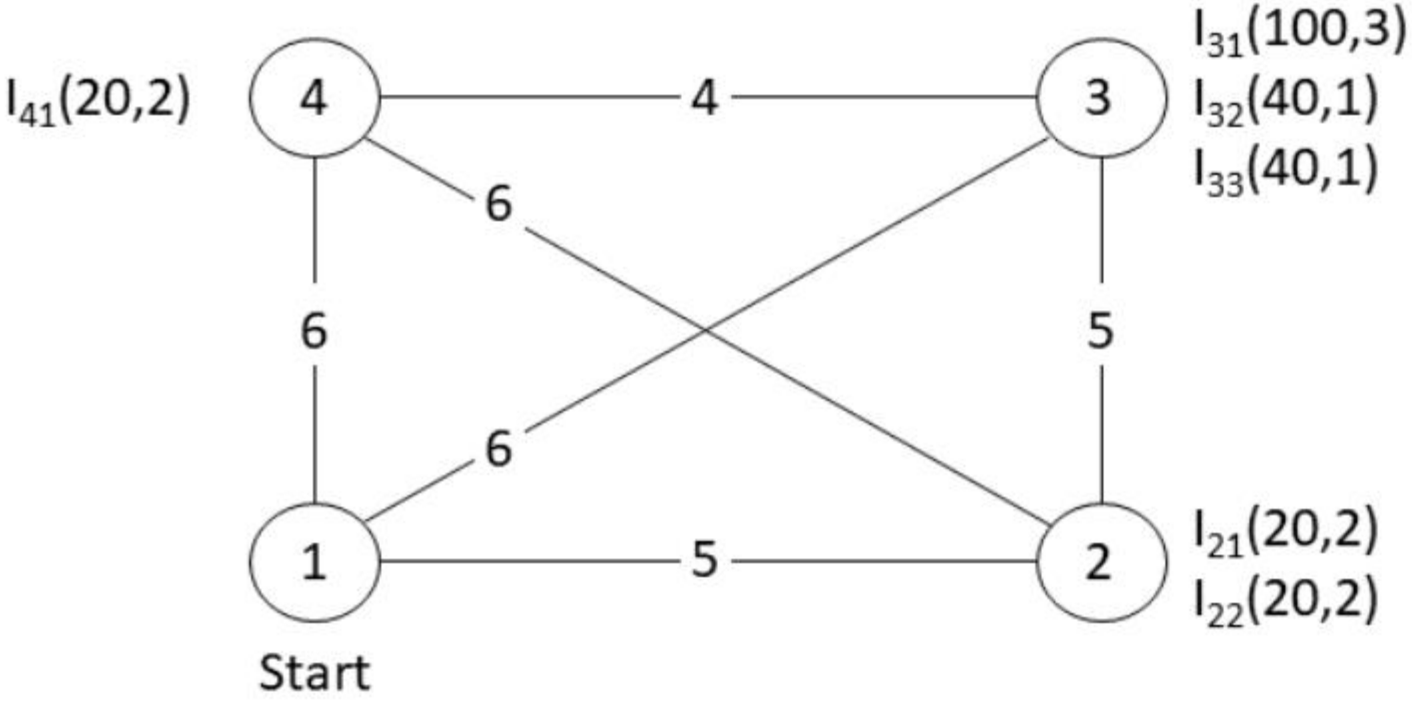
TTP Example

- $z = (0, 0, 0, 1, 1, 0)$
- $g(z) = (20 \cdot 0) + (30 \cdot 0) + (100 \cdot 0) + (40 \cdot 1) + (40 \cdot 1) + (20 \cdot 0) = 80$



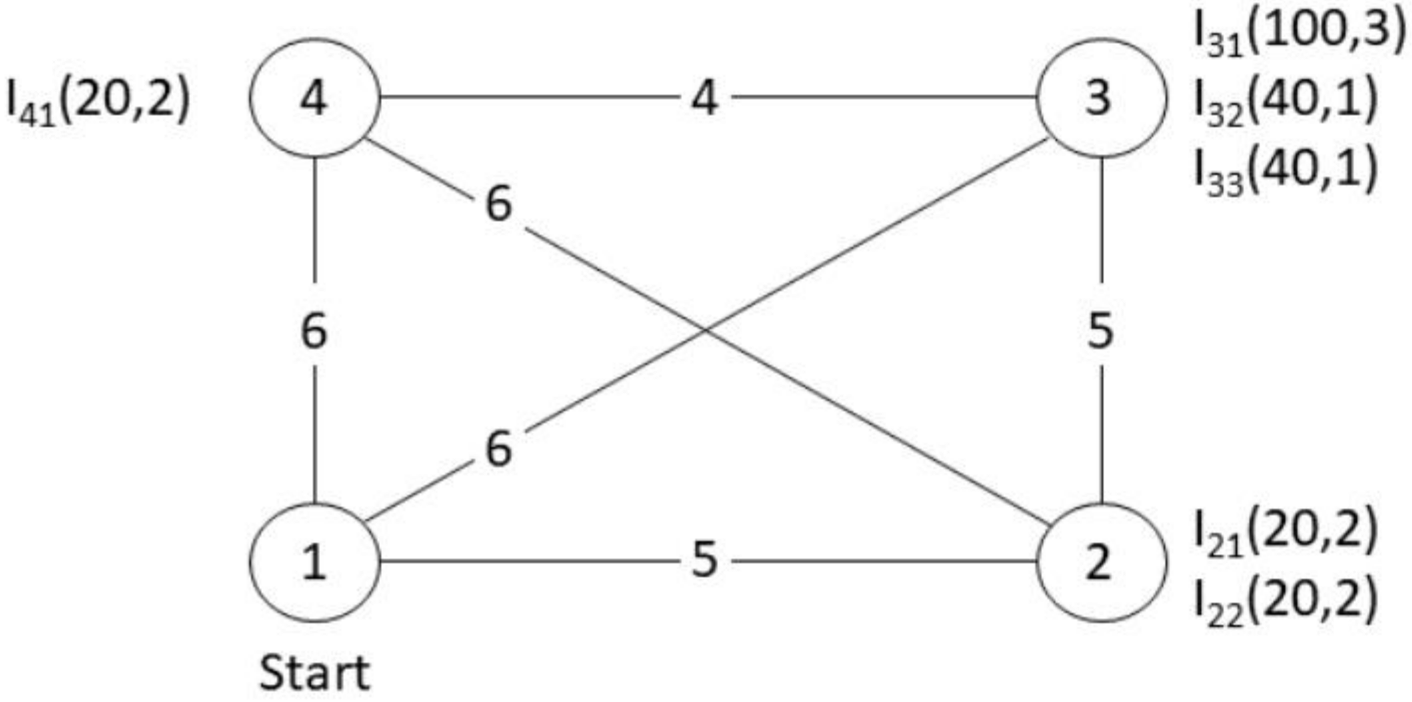
TTP Example

- $x = (1,2,4,3)$
- $f(x,z) = \frac{5}{(1-\frac{1-0.1}{3}\cdot 0)} + \frac{6}{(1-\frac{1-0.1}{3}\cdot 0)} + \frac{4}{(1-\frac{1-0.1}{3}\cdot 0)} + \frac{6}{(1-\frac{1-0.1}{3}\cdot 2)}$
- $f(x,z) = 5 + 6 + 4 + 15 = 30$



TTP Example

- $g(z) = 80$
- $f(x, z) = 30$
- $G(x, z) = g(z) - R \cdot f(x, z) = 80 - (1 \cdot 30) = 50$



SuCo-Dif-MA TTP Instructions

Instruction	Input	Output
num_cities		int
num_items		int
shuffle_tour	tour	tour
shuffle_packing_plan	packing_plan	packing_plan
random_swap_tour	tour	tour
random_swap_packing_plan	packing_plan	packing_plan
add_item	packing_plan, item	packing_plan
remove_item	packing_plan, item	packing_plan
get_item_from_city	tour, packing_plan, int	item
get_city_by_index	int	city
get_next_city_in_tour	tour, city	tour, city
get_city_of_item	item	city
make_weight	packing_plan	packing_plan
swap_cities	city, city, tour	tour
add_random_item	packing_plan	packing_plan
get_random_city		city
get_random_item		item
get_starting_city		city
get_item_weight	item	int
get_item_profit	item	int
get_city_distance	city, city	int
apply_lin_kernighan	tour	tour

Experiments

- Representative benchmark set of community adopted TTP instances ranging from smallest to largest
- TTP Categories
  - Category 1 – 1 item per city, bounded strongly correlated item weight/profit, small knapsack capacity
  - Category 2 – 5 items per city, uncorrelated similar weight/profit, average knapsack capacity
  - Category 3 – 10 items per city, uncorrelated weight/profit, large knapsack capacity

SuCo-Dif-MA TTP Results

AUBURN UNIVERSITY

- Category 1 instance results

Instance	MA2B	SuCo-Dif-MA
eil76	4201 (2.32)	3997 (2.04)
u159	4575 (2.27)	4317 (2.02)
a280	18413 (0.74)	16822 (1.68)
u574	22648 (1.76)	21003 (1.54)
dsj1000	**77942** (0.47)	72111 (1.78)
fl1577	**92801** (0.38)	83011 (1.02)
pcb3038	151339 (1.19)	146904 (1.08)
pla7397	368985 (0.82)	342756 (1.01)
usa13509	**772507** (0.62)	716548 (1.21)
pla33810	**1763677** (0.59)	1592501 (0.98)

SuCo-Dif-MA TTP Results

- Category 2 instance results

Instance	MA2B	SuCo-Dif-MA
eil76	30117 (0.72)	29001 (1.01)
u159	58754 (0.57)	57237 (0.89)
a280	112534 (0.62)	109657 (0.86)
u574	253576 (0.76)	250092 (1.12)
dsj1000	339318 (1.44)	330872 (1.71)
fl1577	651565 (1.07)	639255 (1.21)
pcb3038	**1191961** (0.58)	1062091 (1.34)
pla7397	**4020075** (2.0)	3679271 (1.66)
usa13509	**7556691** (1.35)	6821558 (2.12)
pla33810	**14594137** (1.01)	10051312 (2.09)

SuCo-Dif-MA TTP Results

- Category 3 instance results

Instance	MA2B	SuCo-Dif-MA
eil76	108037 (0.05)	100213 (0.18)
u159	244544 (0.36)	241989 (0.21)
a280	436932 (0.17)	430002 (0.20)
u574	965753 (0.26)	957245 (0.25)
dsj1000	1466106 (0.35)	1410276 (0.39)
fl1577	2589346 (0.71)	2526212 (0.49)
pcb3038	**4627657** (0.56)	4023778 (0.89)
pla7397	**13612483** (1.39)	12098128 (1.01)
usa13509	**25180776** (0.43)	20187221 (0.87)
pla33810	**55622060** (0.29)	45539817 (0.56)

TTP Discussion

AUBURN UNIVERSITY

- Selective optimization
  - Larger impact on TTP
  - Best found approach was to skip a number of primary generations between each support generation (similar to previous work), but execute several support generations in a row
- Evaluations per local optimizer
  - Previous work found the optimal setting for this parameter was small (1 for SuCo-MA and 4 for SuCo-Dif-MA), however on TTP the optimal setting was 7 or 8 depending on the problem instance

TTP Discussion

- Local optimizer programs are very sensitive to change
- Strategies that worked in previous generations may not be effective in the current generation due to changes in the primary population
- A method to aid the support population in offspring generation
  - Which primitives are effective?
  - What combinations of primitives are effective?

TTP Conclusions

- SuCo-Dif-MA can be applied to TTP with minimal changes (new instruction set)
- SuCo-Dif-MA is a promising approach producing competitive results
- More work is required to solve some underlying problems but could result in performance improvements

Final Conclusions

- SuCo has been used to successfully evolve mutation step size, crossover operators, and local optimization operators
  - SuCo can improve performance when compared to both static parameters and operators as well as self-adapted parameters
- SuCo-MA and SuCo-Dif-MA evolved mainly stochastic local optimizers
  - Creating high quality deterministic optimizers is probably much harder than simple, random, “lucky” operators
  - It is challenging to create deterministic strategies instead of informed mutators

Future Work

- Evolve more operators and parameters
  - Selection operators, population size, etc.
- Perform experiments with more than two support populations
- Determine if performance improvements from diffusion model benefits other operators/parameters in SuCo
- Improve upon local optimizer evolution
  - General improvements to program evolution to address stability/sensitivity issues
  - Address challenges of online dynamic program evolution

Questions

AUBURN UNIVERSITY

- Thank you!
- Questions?