情報処理学会論文誌 Vol.54 No.3 1141–1152 (Mar. 2013)

ユーザの体力変化に対応可能な観光スケジュールの立案手法

武 兵[1,a)] 孫 為華[1,b)] 村田 佳洋[2,c)] 安本 慶一[1,d)] 伊藤 実[1,e)]

受付日 2012年3月23日, 採録日 2012年12月7日

概要：観光においては，ユーザの好む観光スポットをより多く回るスケジュールを立案することが望ましい．しかし各観光スポットにつき，観光方式や観光時間によって必要な体力が異なり，ユーザの体力がスケジュールを遂行できない場合がある．本論文では，観光中に休憩を適宜に行うことで体力の範囲内で最も満足度が高くなる観光スケジュールを求める問題を取り扱う．本問題は NP 困難であり，問題例の規模が大きいときには，実用時間で最適解を算出することは困難である．実用時間で準最適解を得るため，ヒューリスティックな探索法である捕食法に基づいて複数の観光スポットを回る休憩なしのスケジュールを求めたうえ，局所探索を用いて適宜に休憩を差し挟むことで解を求める．提案手法を評価するため，異なる観光地候補数を有する複数のインスタンスを用いてシミュレーション実験を行った．その結果，候補数 10 の場合，提案手法は全探索で得られた解の 95.65%の満足度を有するスケジュールを 13 秒で得られることを確認した．

キーワード：ナビゲーションシステム，広域・近傍探索，巡回スケジューリング，組合せ最適化アルゴリズム

Stamina-Aware Sightseeing Tour Scheduling Method

Bing Wu[1,a)] Weihua Sun[1,b)] Yoshihiro Murata[2,c)] Keiichi Yasumoto[1,d)] Minoru Ito[1,e)]

Received: March 23, 2012, Accepted: December 7, 2012

Abstract: Tour schedules are required to include multiple sightseeing spots taking into account the user's preference, but the stamina of tourists may be depleted during sightseeing. In this paper, we formulate the sightseeing scheduling problem to maximize the user's satisfaction taking stamina into account. In this problem, break times are allocated in schedules to hold constraint of stamina. This problem is NP-hard, and thus it is difficult to be solved in practical time. In order to obtain a semi-optimal solution in practical time, we propose a method that derives a schedule visiting multiple sightseeing spots with no break times based on a predatory search technique and then allocates the break times in the schedule using a local search technique. To evaluate the proposed method, we compared our method with conventional methods through computer simulations for several different instances containing 10 sightseeing spots. As a result, the proposed method composed the schedule whose expected satisfaction is 95.65% of the optimum solution in 13 sec.

Keywords: navigation system, extensiveand area-restricted search, travelscheduling, combinatory optimization algorithm

1. はじめに

近年，ユーザに対し観光地の各スポット（以降，目的地と呼ぶ）への経路案内や周辺情報の提供を行う様々なパーソナルナビゲーションシステムが提案されている．パーソナルナビゲーションシステムの主な用途として観光への利用があげられる．ユーザのコンテキスト（位置，時間，周辺環境など）に応じて適切なルートを動的に案内す

[1] 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[2] 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University, Hiroshima 731–3194, Japan
a) bing-w@is.naist.jp
b) sunweihua@is.naist.jp
c) yosihi-m@hiroshima-cu.ac.jp
d) yasumoto@is.naist.jp
e) ito@is.naist.jp

るシステム [1], [2] や屋内施設におけるルート案内 [3]，携帯端末に対し観光案内などの情報を提供するためのシステム [4] がある．また，観光中のユーザ支援を目的に，多様な観光計画や，交通機関乗り換えに関する情報の提供，屋内外の案内のための統合アプリケーションが提案されている [5]．観光では，複数の観光地（目的地）を対象とし，中から満足度の高い経路をたどることを目的とする．観光問題は一般的に巡回セールスマン問題と見なし，この問題を効率良く解くヒューリスティックアルゴリズムとして，Lin-Kernighan 法 [6]，焼きなまし法（Simulated Annealing）など，厳密に最適解を求めないことで計算時間を短くする方法がある．より複雑な定義の問題を扱う解法としては，遺伝的アルゴリズム [7]，捕食法（Predatory Search: PS 法）[8], [9], [10]，大規模局所探索法 [11]，また Consultant-Guided 法 [12], [13] といった厳密解を保証しないヒューリスティックアルゴリズムを用いることも多い．しかし，これらは基本的に時間制約内で複数の目的地を巡回するスケジュールを計算するものであり，観光者の体力を考慮して，嗜好に合ったスケジュールを立案する機能を持たない．

我々の研究グループは，観光のためのパーソナルナビゲーションシステム P-Tour [14] を提案している．P-Tour は，ユーザの希望に応じた複数目的地の巡回スケジュールを立案し，また各目的地における案内機能を提供する．また，P-Tour 上に天気変化を考慮した観光案内機能 [15] を追加するなど，現実環境に適用する拡張を行っている．しかし，このシステムは，観光者の体力と観光方式に応じて目的地の満足度が変化する問題を取り扱っていない．観光旅行においては，体力が低下すると満足度が急激に下がるため，観光者の体力にあわせた無理のないスケジュールを作成することが必要である．観光者の体力を考慮した観光方式としては，登山ガイドブックに表示されるコースごとの必要体力の案内や，旅行会社による対象年齢別の旅行プランの推奨などがある．たとえば，ケーブルカーで登山する場合，体力の消耗度が少なくなるが，得られる満足度も歩行登山の場合と異なる．体力を考慮した場合，どこでいつ休憩を入れるかのスケジューリングが重要となる．体力と時間の制約条件の下で適宜に休憩を入れ，満足度を最大化するような観光スケジューリングは複雑な最適化問題である．

本論文では，観光者の体力と時間を制約とし，適宜に休憩をとりながら，複数の目的地に対し，好きな方式で巡るスケジュールの立案手法を提案する．本手法では，観光候補地を異なる方式で観光する場合の体力消費と得られる満足度をあらかじめ定義し，スケジュール全体で得られる満足度をできるだけ大きくすることを目的とする．観光候補地から選択する複数の目的地を回る最も満足度の高いスケジュールを算出するためには，観光中に適宜に休憩を挟むことで，体力残がマイナスにならないようにすることが有効である．提案手法では，このことに着目し，捕食法を用いて，休憩なしのスケジュールをまず求め，休憩を差し挟む回数およびタイミングを近傍探索で求めるというアプローチを採用する．提案手法により得られた解の最適性を評価するため，10 観光候補地を有するインスタンスに対し提案手法と全探索で求めた解を比較した．その結果，最適解の 95.65%以上の満足度を持つ解を平均 12.99 秒で得ることができた．また，欲張り法と比較したところ，20 カ所の観光候補地を持つインスタンスにおいて 1.36 倍の満足度を持つ解を得ることができた．

2. 体力を考慮した最大満足度観光スケジューリング問題

本章では，体力を考慮した最大満足度の観光スケジュールを算出する問題を定義する．

2.1 問題概要

観光スケジュールを立案するために，まず，観光者と観光候補地の関係をモデル化する．

表 1 に観光候補地および観光方式の例をあげる．便宜上，消耗体力値は負の値として定義している．たとえば，$q_1 = -110$，$q_3 = -340$ のように，同じ観光候補地でも観光方式が違うと，満足度および観光者の体力消耗の度合いが変化する．本問題では，観光者の体力が許す範囲で満足度が最大になる観光スケジュールを求めることを目的とする．**図 1** はスケジュール例である．出発時間を 9 時，帰着時間を 15 時とする．観光者に対して一時的に算出され

表 1 観光候補地に対する観光方式の一例
Table 1 An example of the sightseeing spot and its methods.

観光候補地	観光候補地 v_1			観光候補地 v_2		観光候補地 v_3	
観光方式	q_1	q_2	q_3	q_4	q_5	q_6	q_7
消耗体力値	−110	−200	−340	−500	−700	−120	−90
滞在時間（hour）	1	1.5	2	1.5	2	1.5	1
満足度	40	60	75	60	80	90	70

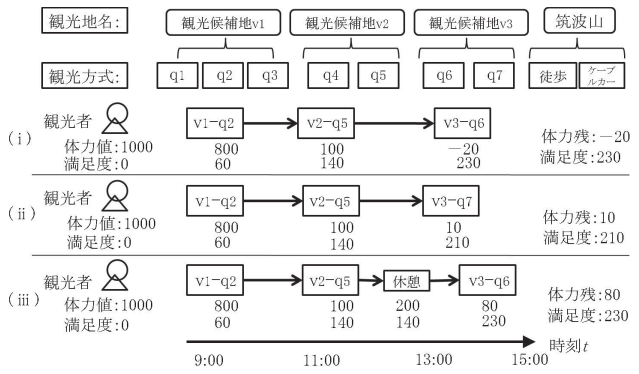

図 1 スケジュールの例
Fig. 1 Example of schedules.

表 2 提案手法で求めた観光スケジュールの例
Table 2 An example of schedules obtained by proposed method.

到着時間	観光候補地	観光方式	満足度	体力値	出発時間	帰着時間
	Hotel	By Car	0.0	900.0	09:00	
09:04	Spot7	method1	101.4	644.0	10:34	
10:43	Spot5	method2	98.7	303.0	11:43	
11:43	Spot5	have rest	0.0	343.0	11:47	
12:00	Spot1	method1	93.1	10.0	13:00	
13:00	Spot1	have rest	0.0	210.0	13:20	
13:24	Spot10	method1	105.3	5.0	14:54	14:58
観光スケジュールの満足度：398.5						

た観光スケジュールを $v_1 \rightarrow v_2 \rightarrow v_3$ とする．観光候補地 v_1，v_2 に対し，観光方式 q_2，q_5 を選択し，その後，観光候補地 v_3 の観光方式として q_6（消耗体力値が -120）を選択すると（図 1 (i)），体力値が負になるため，このようなスケジュールは実行不可能である．次に，図 1 (ii) のように，観光候補地 v_3 の観光方式を q_7（消耗体力値が -90）に変更すると，体力内での観光は可能になるが，満足度は低くなる．一方，図 1 (iii) のように，観光候補地 v_2 と v_3 の間に休憩を入れると，体力値が回復するため，体力内でかつ満足度の高い観光が可能になる．この場合，スケジュールの満足度は 230 となる．本問題の目的は，観光者の体力に関する制約条件を満たしつつ，スケジュールの満足度を最大化することである．休憩が必要な場合，観光後に同じ場所で休憩を行うよう設定する．**表 2** は，提案手法で求めた休憩を含むスケジュールの一例である．例では，出発時間を 9 時，帰着時間を 15 時としている．体力値が 0 以下にならないように，適宜休憩が入っていることが分かる．

2.2 問題の定式化

本問題ではまず，2 つの集合 V，Q と関数 $dist(v_1, v_2)$ が与えられる．V は観光候補地（例：法隆寺，奈良公園など）の集合を，Q は観光方式の集合を表す．また，$Q_{v_i} \subseteq Q$ は観光候補地 $v_i \in V$ に対応した観光方式の部分集合を表す．$dist(v_1, v_2)$ は観光候補地 $v_1, v_2 \in V$ の間の距離を表す．スケジュール S を以下の式で定義する．

$$S = [d_0, ..., d_n]$$

ここで，d_j は j 番目に観光する観光候補地における観光方式と休憩時間を表しており，$d_j.v$，$d_j.q$，$d_j.r$ はそれぞれ観光地名，観光方式と観光後の休憩時間を示すこととする．

スケジュール S の満足度を与える関数 $Sat(S)$ を以下の式で定義する．ただし，$term1$ はスケジュールで得られる総満足度，$term2$ はホテル $(d_0.v)$ から出発し，各観光地 $(d_j.v)$ に訪問し，最後に，出発地点のホテル $(d_0.v)$ へ戻る移動距離に対するペナルティである．

$$Sat(S) = \underbrace{\sum_{j=0}^{n} d_j.q.sat}_{term1} \underbrace{-\alpha \left(\sum_{j=0}^{n-1} dist(d_j.v, d_{j+1}.v) + dist(d_n.v, d_0.v) \right)}_{term2} \quad (1)$$

ここで $q.sat$ は観光方式 q の満足度，n は観光スケジュール内の観光候補地の数である．また，α は観光経路の短さ（無駄な経路の少なさ）をどの程度重視するかを示すパラメータである．つまり，移動距離 1 km ごとに満足度は α 点ずつ減点される．

ユーザが目的地 d_j での観光終了直後に残っている体力値 $RemSt(d_j)$ は以下の式 (2) で与えられる．ただし，体力値はユーザの最大体力値 $MaxST$ を超えない．$InitSt$ は出発時の体力値である．

$$RemSt(d_j) = \begin{cases} \mathrm{Min}(SumSt(d_j), MaxSt) & \text{if } j > 0 \\ InitSt & \text{otherwise} \end{cases} \quad (2)$$

$SumSt(d_j)$ は目的地 d_{j-1} での体力値と目的地 d_j での体力消耗値，回復する体力値の総和であり，以下の式 (3) で与えられる．

$$SumSt(d_j) = RemSt(d_{j-1}) + d_j.q.conp + RstSt(d_{j-1}) \quad (3)$$

ここで，$q.conp$ は観光方式 q の体力消耗値である．また，目的地 d_j の観光後の休憩により，回復する体力値 $RstSt(d_{j-1})$ は以下の式 (4) で与えられる．$habitus$ は単位時間内に回復する体力値を表す．つまり，休憩時間 1 分ごとに体力値は $habitus$ 分回復する．

$$RstSt(d_{j-1}) = habitus \cdot d_{j-1}.r \quad (4)$$

スケジュールは以下の制約を満たさなければならない．

観光の間，ユーザの体力は負になってはならない．全目的地におけるユーザの体力に関する制約条件を式 (5) で示す．

$$\forall d_j.v \in [0, n-1], d_{j+1}.q.conp \leq RemSt(d_j) \qquad (5)$$

スケジュール S における観光時間制約条件は式 (6) に示す．ただし，$term3$ はスケジュールで得られる総滞在時間，$term4$ はスケジュールで得られる総休憩時間，$term5$ は各観光候補地間の移動時間，$term6$ はホテルに戻るのにかかる移動時間である．

$$MaxTi \geq \underbrace{\sum_{j=0}^{n} d_j.q.stayt}_{term3} + \underbrace{\sum_{j=0}^{n} d_j.r}_{term4}$$
$$\underbrace{+\sum_{j=0}^{n-1} \frac{dist(d_j.v, d_{j+1}.v)}{speed}}_{term5} + \underbrace{\frac{dist(d_n.v, d_0.v)}{speed}}_{term6} \qquad (6)$$

ここでは，$q.stayt$ は観光方式 q の滞在時間であり，$speed$ は 2 つのスポット間の移動スピードであり，$MaxTi$ は最大観光時間である．

以上をふまえて本問題の目的関数を以下の式で定義する．

$$\begin{aligned} &\text{maximize } Sat(S) \\ &\text{subject to } restrictions(5) - (6) \end{aligned} \qquad (7)$$

2.3 NP 困難性

経路探索問題として一般化された巡回セールスマン問題（GTSP）がある．これは巡回セールスマン問題（TSP）において，すべての都市がいくつかのクラスタに分けられたもので，この GTSP を TSP に変換する手法 [16] も提案されている．またこの問題を効率良く解くため，いくつかのヒューリスティックアルゴリズム [12], [13] が提案されている．

提案する問題は，一般化された巡回セールスマン問題（GTSP）に出発/帰着点や時間制約，体力制約などを加えたものであり，GTSP を本問題に帰着することができる．GTSP は NP 困難なので [16]，提案する問題も NP 困難である．

3. 体力を考慮した観光スケジューリングアルゴリズム

2 章で定義した問題は NP 困難であるため，問題例の規模が大きいときには，実用時間内で最適解を求めることは難しい．そこで，捕食法を利用したヒューリスティックアルゴリズムを提案する．本章では，まず 3.1 節においてヒューリスティック手法である捕食法を紹介し，次に 3.2 節においてこれを利用した提案手法について述べる．

3.1 捕食法

自然界の捕食獣は捕食行動が驚くほど類似している．捕食獣が捕食するとき，複数の狩場をターゲットとし，順番に探索し，獲物が見つかるまで狩場をすばやく切り替える．獲物を発見すると，その狩場の最も獲物の多い場所に近づき，狩りやすい獲物を選択し，これを追って捕獲する．獲物を逃した場合，次の狩場に変更する．動物の捕食行動を模した空間探索手法である捕食法は，Linhares が 1998 年に提案した [8]．**図 2** に示すように，捕食法のアルゴリズムは次の 2 つの探索過程を含む：(1) 広域探索–最初に全体の解空間内で，制約条件を満たす解のうち，暫定解を 1 つ求める．指定された繰返し規定回数で暫定解が改善できない場合，アルゴリズムは終了する：(2) 近傍探索–広域探索で見つかった暫定解に基づき近傍範囲を設定したうえで探索し，暫定解より良い解の有無を確認する．良い解を発見した場合，近傍探索の範囲を新たに設定し，探索を繰り返す．設けた近傍探索範囲内で良い解が見つからなければ，近傍探索を放棄し，広域探索に戻る．これらの 2 つの探索は解を改善するため，交互に繰り返し適用される．近傍の大きさを調整することにより，広域探索と近傍探索のバランスを調整する点に特色がある．

Linhares は巡回セールスマン問題（TSP）[8] および大規模集積回路（VLSI）設計問題 [9] に捕食法を適用した．また，Liu らは解の近傍を再定義することにより，捕食法を改良した [10]．本論文では，Linhares のオリジナルの捕食法 [8], [9] を改良し，2 章で定義した観光スケジューリング問題に適用する．

3.2 提案アルゴリズム

対象とする観光スケジューリング問題は，観光候補地，観光順，観光方式など，選択の対象が段階的に絞られるという特徴を持っている．本問題の解である観光スケジュールでは，各目的地 d を 4 項組 $d = \langle v, j, q, r \rangle$ で表す．ここで $v \in V$ は観光候補地，$j \in [0, n]$ はその観光候補地 v の巡回順序，$q \in Q$ は観光方式であり，また，r は観光後に

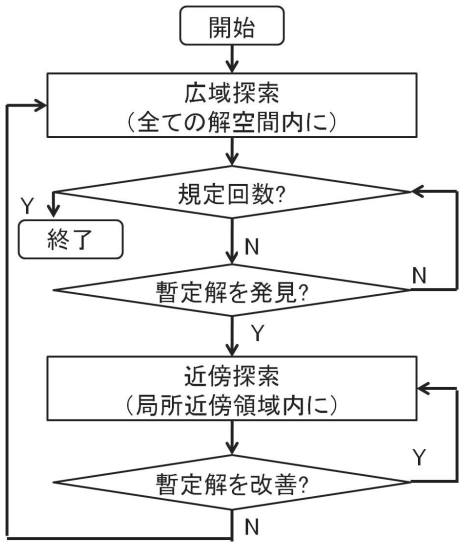

図 2 捕食アルゴリズム
Fig. 2 Predatory Search Algorithm.

とる休憩時間である．たとえば，$\langle v_1, 2, q_2, 4\rangle, \langle v_2, 1, q_1, 0\rangle$ は，1 番目に観光候補地 v_2 を観光方式 q_1 を用いて観光してから，2 番目に観光候補地 v_1 を観光方式 q_2 を用いて観光し，4 分休憩するようなスケジュールを示す．観光順序と観光方式の決まっていない目的地は $\langle v_2, *, *, r\rangle$ のように記述する．

提案するアルゴリズムは（1）暫定解算出フェイズ，（2）近傍探索フェイズの 2 つのフェイズで構成される．まず暫定解算出フェイズ（捕食法の広域探索に相当）では，ランダムに生成した解を通常の局所探索により改善し，暫定解を得る．次に近傍探索フェイズ（捕食法の近傍探索に相当）では，ランダムに複数の部分空間を生成し，暫定解に近い部分空間から順に探索していく．暫定解よりも良い解が見つかった場合，その解を中心に部分空間を再生成し，改めて探索を行う．この過程は再帰的に適用され，有望な暫定解（狩場）の近傍をあたかも捕食獣が獲物を探すかのように探索することができる．全体的なアルゴリズムの擬似コードを Algorithm1 に示す．スケジュール S^{cm} と S はそれぞれ暫定解，最終解である．また，関数 $TemporarySolutionSearchPhase()$ は暫定解算出フェイズ，関数 $NeighborhoodSearchPhase()$ は近傍探索フェイズである．

Algorithm 1 *The Proposed Algorithm*

```
1: S^cm ← TemporarySolutionSearchPhase()
2: S ← NeighborhoodSearchPhase(S^cm)
```

以下，これらフェイズについて詳しく述べる．

3.2.1 暫定解算出フェイズ

巡回セールスマン問題では，すべての目的地を巡回する経路を求める．しかし本論文で取り扱う問題では，時間制約に基づいて巡回する目的地をいくつか選択しなければならない．従来の捕食法ではランダムに生成された初期解から探索を始めるが，提案手法では探索効率を高めるため，まず欲張り法と局所探索法を利用して質の高い初期解（暫定解）を算出する．このフェイズの擬似コードを Algorithm2 に示す．

Algorithm 2 *TemporarySolutionSearchPhase*

```
 1: n ← MaxTi/min_{q∈Q}{q.stayt}, δ ← Maxloc
 2: V' ← V, O ← {1, ..., n}, S^c ← ∅, S^cm ← ∅
 3: while |S^c| < n do
 4:     select v ∈ V' at random
 5:     select j ∈ O at random
 6:     q ← argmax_{q∈Q_v} PerCostSat(q)
 7:     S^c ← S^c ∪ {⟨v, j, q, *⟩}
 8:     V' ← V' − {v}, O ← O − {j}
 9: end while
10: S^cm ← Modify(S^c)
11: S^cm ← LocalSearch(S^c, S^cm, δ)
12: return S^cm
```

Algorithm2 では，まず，観光時間制約条件のもとで訪れることが可能な目的地の最大数を求め，候補地からランダムに目的地を選択し，1 つの観光スケジュールを生成する．次に，関数 $PerCostSat()$ を利用し，欲張り法で各目的地の観光方式を決める．それから，関数 $Modify()$ を適用し，適切に休憩を入れ，観光方式を含んでいる観光スケジュールの満足度を算出する．最後に，局所探索関数 $LocalSearch()$ を用いて解を改善する．

以下に擬似コードを用いて詳細に述べる．1 行目で巡回する目的地の数 n を計算する．ここでは，最大観光時間 $MaxTi$ および最小の観光時間 $q.stayt$ から，巡回しうる最大の目的地数を求める．定数 δ は局所探索の回数パラメータであり，予備実験により，$Maxloc = 5 \cdot |V|$ の値を用いている（後述）．2 行目では，観光候補地の集合 V'，観光順の集合 O，およびスケジュール比較解 S^c と暫定解 S^{cm} を初期化する．3–9 行目では，巡回する観光候補地 $v \in V'$，観光順 $j \in O$ をランダムに決定し，目的地 $\langle v, j, q, *\rangle$ を構成する．6 行目で，欲張り値（式 (8) で後述）を計算し，最も高い欲張り値を持つ観光方式 q を，その観光候補地 v の観光方式とする．7–8 行目では，生成された目的地を S^c に追加する．S^c に属する目的地の数が n になるまで，上記のプロセスを繰り返す．また，8 行目では，任意の 2 つの目的地が同じ観光地や同じ観光順を含まないよう制約条件を定義する．たとえば，観光候補地の集合 $|V| = 6$，巡回する目的地の数 $n = 4$ の場合，ランダムに生成する 1 つのスケジュールは $\{\langle v_0, 0, *, *\rangle\ \langle v_1, 2, q_2, *\rangle\ \langle v_2, 1, q_1, *\rangle\ \langle v_5, 4, q_1, *\rangle\ \langle v_3, 3, q_4, *\rangle\}$ となる．ここでは，出発/帰着ホテルは $\langle v_0, 0, *, *\rangle$ とする．10 行目では，目的地の休憩時間 r の追加や，制約時間を超える目的地の削除などの処理を手続き $Modify()$ を利用し，S^c を修正する．最後に，S^c を改善するため，局所探索法 $LocalSearch()$ を用いる．関数 $Modify()$ と関数 $LocalSearch()$ の説明は後述する．

以降，Algorithm2 内で用いる関数について述べる．

3.2.1.1 $PerCostSat(q)$ 関数

関数 $PerCostSat(q)$ は，観光方式 q ごとに与えられる“どれだけ効率的に満足度が得られるか（単位時間あたりの満足度）”の指標である．この値が大きいほど，観光地としての価値が高い．式 (8) でこの値の計算方式を示す．ただし，$q.conp/habitus$ は消耗体力に対する回復時間を表す．すなわち，$PerCostSat(q)$ の値に対し，時間と消耗体力が影響を与え，滞在時間が長い，また消耗体力の大きい観光地は価値が低い．一方，ユーザの体力回復能力が“強ければ”，消耗体力の影響が小さくなる．

$$PerCostSat(q) = q.sat/(q.stayt + abs(q.conp)/habitus) \quad (8)$$

たとえば，表 1 では，観光候補地 v_1 の観光方式 q_2 に対し，消耗体力値 $q_2.conp$ は -200，滞在時間 $q_2.stayt$ は

$1.5hour$，満足度 $q_2.sat$ は 60 である．もし観光者の休憩状態において毎分体力回復率 $habitus = 10$ と設定すると，単位時間あたりの満足度の結果は $PerCostSat(q_2) = 60/(1.5 * 60 + 200/10) = 0.55$ である．

3.2.1.2 $Modify(S^c)$ 関数

関数 $Modify(S^c)$ は，スケジュール S^c が制約条件を満たすよう修正して返す関数である．体力の制約条件 (5) を満たすため，この関数は，スケジュール S^c の中に適切に休憩を入れる．j 番目に巡回する目的地 d_j で体力値が足りない場合，直前の目的地 d_{j-1} の休憩時間を以下の式で計算する．

$$d_{j-1}.r = -RemSt(d_j)/habitus$$

たとえば，表 2 では，3 番目の観光候補地 $Spot9$ に巡回すると，体力値が足りなくなるため，直前の観光候補地 $Spot1$ において 4 分の休憩を行う．またスケジュールが帰着時間の制約 (6) を満たさない場合，満たすまで最後から 1 つずつ目的地を削除してゆく．この関数は，以上の処理によって修正されたスケジュールを返す．

3.2.1.3 $LocalSearch(S^c, S^{cm}, \delta)$ 関数

関数 $LocalSearch(S^c, S^{cm}, \delta)$ は，スケジュール S^c 内の観光候補地をランダムに交換する局所探索を行う関数である．スケジュール内の目的地 d_i をランダムに選択し，また観光候補地 v' を V からランダムに選択する．選択された観光候補地 v' がスケジュール S^c 内の別の目的地 d_j に含まれていたときは，目的地 d_i と d_j の観光順を交換し，新たなスケジュール S'^c を得る．選択された観光候補地 v' が含まれていなかったときには，目的地 d_i の観光候補地を v' で上書きする．たとえば，スケジュール $\{\langle v_0, 0, *, *\rangle\ \langle v_1, 2, q_2, *\rangle\ \langle v_2, 1, q_1, *\rangle\ \langle v_5, 4, q_1, *\rangle\ \langle v_3, 3, q_4, *\rangle\}$ 内の目的地 $\langle v_2, 1, q_1, *\rangle$ をランダムに選択し，また観光候補地 $\langle v_5, *, *, *\rangle$ を V からランダムに選択する．選択された観光候補地 $\langle v_5, *, *, *\rangle$ がスケジュール内に含まれているので，2 つの目的地の観光順を交換し，スケジュール $\{\langle v_0, 0, *, *\rangle\ \langle v_1, 2, q_2, *\rangle\ \langle v_2, 4, q_1, *\rangle\ \langle v_5, 1, q_1, *\rangle\ \langle v_3, 3, q_4, *\rangle\}$ を得る．もし観光候補地 $\langle v_6, *, *, *\rangle$ が選択された場合，スケジュール $\{\langle v_0, 0, *, *\rangle\ \langle v_1, 2, q_2, *\rangle\ \langle v_6, 1, *, *\rangle\ \langle v_5, 4, q_1, *\rangle\ \langle v_3, 3, q_4, *\rangle\}$ を得る．また Algorithm2 の 6 行目と同様の方法を用い欲張り値が最大の観光方式を与えて，新たなスケジュール S'^c を得る．スケジュール S'^c に前述の関数 $Modify(S'^c)$ を適用し，制約条件を満たすよう修正する．最後に変更前のスケジュール S^{cm} と変更されたスケジュール S'^c の満足度を比較し，改善されていれば，採用する．でなければ，破棄する．この改善の試みは δ 回数繰り返す．

3.2.2 近傍探索フェイズ

提案手法では捕食法を利用し，観光スケジュールを観光候補地，観光順，観光方式の順番で決定する．ここではスケジュールの解空間を全体解空間，部分解空間，部分部分解空間，そして解の順で細分化してゆく．

- **全体解空間**：スケジュールの観光候補地，観光順，観光方式，休憩時間をすべて未定とした解空間であり，$\langle *, *, *, *\rangle^n$ で表す．
- **部分解空間**：巡回する観光候補地のみを決定したもの．その他（順番や観光方式）を未定とした解空間であり，$\langle v, *, *, *\rangle^n$ で表す．
- **部分部分解空間**：ある部分解空間でさらに，観光順番を決定したもの．観光方式はいまだ未定であり，$\langle v, j, *, *\rangle^n$ で表す．
- **解**：ある部分部分解空間でさらに，観光方式と休憩時間を決定したもの．これは**観光スケジュール**，つまり 1 つの解であり，満足度が計算できる．

各解空間の探索方法をそれぞれ，Algorithm3，4，6 に示す．

Algorithm3 は，全体の解空間の探索を行うアルゴリズムである．暫定解算出フェイズで得られた解 S^{cm} を暫定解 S の初期値とする．観光候補地のみ含む複数の部分解空間をランダムに作成し，暫定解 S に属する空間と比較する．関数 $DistV()$ を適用し，比較結果として，複数の部分解空間をソートする．次に，各部分解空間内を順に探索する関数 $SubDomainSearch()$ を繰り返し呼び出して，得られた解が改善された場合には，暫定解 S を更新し，全部分解空間を再生成し，再度探索をする．生成されたすべての部分解空間を探索しても暫定解 S が更新されなかったとき，アルゴリズム全体を終了する．

Algorithm 3 *NeighborhoodSearchPhase*

```
1: S ← S^cm, n ← MaxTi/min_{q∈Q}{q.stayt}, l_max ← β, D_k ← ∅, SearchDone ← false
2: while SearchDone = false do
3:    for all k such that 1 ≤ k ≤ l_max do
4:       generate D_k by selecting n v ∈ V at random
5:       D_k.distv ← DistV(S, D_k)
6:    end for
7:    generate D'_1, ..., D'_{l_max} by sorting D_1, ..., D_{l_max} in the increasing order of D_k.distv (1 ≤ k ≤ l_max)
8:    k ← 1, UpDated ← false
9:    while k ≤ l_max and UpDated = false do
10:      S' ← SubDomainSearch(D'_k, n)
11:      if Sat(S') > Sat(S) then
12:         S ← S', UpDated ← true
13:      else
14:         k ← k + 1
15:      end if
16:   end while
17:   if k = l_max + 1 then SearchDone ← true, and give up the neighborhood search
18: end while
19: return S
```

以下に擬似コードを用いて詳細に述べる．3–6 行目で，

l_{max} 個の部分解空間をランダムに作成している．ここでは $D_1, ..., D_{l_{max}}$ で表す．実験では，β（$= l_{max}$）の値は文献 [10] と [17] を参考に行った予備実験より $3n$ としている．部分解空間 D_k は巡回する n 個の目的地の観光候補地のみ含む．4 行目で，観光候補地のみからなる目的地をランダムに D_k に追加していく．具体的には，観光候補地集合 V からランダムに選択し，観光候補地のみからなる目的地を部分解空間に追加し，選択された観光候補地 v を観光候補地列から削除している．5 行目で，暫定解 S に近い部分解空間から探索するため，それぞれの部分解空間と，S に属する空間（観光候補地のみ）との間の距離を関数 $DistV()$ で計測する．暫定解 S に属する空間と部分解空間の距離は相似程度を表す．たとえば，暫定解算出フェイズで得られた暫定スケジュール S= $\{\langle v_0, 0, *, *\rangle \langle v_1, 2, q_2, 0\rangle \langle v_2, 1, q_1, 0\rangle \langle v_5, 4, q_1, 5\rangle \langle v_3, 3, q_4, 10\rangle\}$ とする．$l_{max}= 3 \times 4$ 個の部分解空間 $D_1 = \{\langle v_0, 0, *, *\rangle \langle v_1, *, *, *\rangle \langle v_2, *, *, *\rangle \langle v_3, *, *, *\rangle \langle v_5, *, *, *\rangle\}, ..., D_{12} = \{\langle v_0, 0, *, *\rangle \langle v_3, *, *, *\rangle \langle v_4, *, *, *\rangle \langle v_5, *, *, *\rangle \langle v_6, *, *, *\rangle\}$ をランダムに作成する．それぞれの部分解空間と，S に属する空間との間の距離は $D_1.distv = 0, ..., D_{12}.distv = 2$ である．関数 $DistV()$ の説明は後述する．7 行目で，計測された距離に応じ，距離の近いものから部分解空間をソートし，$D'_1, ..., D'_{l_{max}}$ とする．8–16 行目で，関数 $SubDomainSearch()$ を用いて，部分解空間内を順に探索していく．得られた更新解を S' とする．11–12 行目で，得られた解 S' が改善された場合には，暫定解 S を更新する．そして 3 行目に戻って全部分解空間を再生成し，更新された暫定解 S に近い部分空間から再度探索を開始する．最後の部分解空間まで探索しても改善されなければアルゴリズムは終了する．この時点での S が，全体アルゴリズムにより得られた解となる．

以降，Algorithm3 内で用いる関数について述べる．

3.2.2.1 *SubDomainSearch* 関数

関数 $SubDomainSearch(D, n)$ は，部分解空間 D 内の探索を行う関数である．この関数の擬似コードを Algorithm4 に示す．ここで n は，スケジュールに含まれる目的地の個数である．Algorithm4 では，まず，Algorithm3 で生成した部分解空間内に，関数 $TemporarySolution()$ を適用し，この部分解空間の暫定解 S を生成する．次に，観光候補地の観光順番を決定する複数の部分部分解空間をランダムに作成し，暫定解 S に属する観光順空間と比較する．関数 $SDistV()$ を適用し，Algorithm3 と同様な探索方法で複数の部分部分解空間をソートする．それから，各部分部分解空間内を順に探索する関数 $RandomSelectionInSubSubDomain()$ を繰り返し呼び出す．ここで得られた解が改善された場合には，暫定解 S を更新し，全部分部分解空間を再生成し，再度探索をする．生成されたすべての部分部分解空間を探索しても暫定解 S が更新されなかったとき，Algorithm4 を終了する．

Algorithm 4 *SubDomainSearch Function*

```
1: S ← ∅, O ← {1, ..., n}, m_max ← γ, SD_k ← ∅, SearchDone ← false
2: S ← TemporarySolution(D, n)
3: while SearchDone = false do
4:    for all k such that 1 ≤ k ≤ m_max do
5:       generate SD_k by selecting v ∈ {v'|⟨v', *, *, *⟩ ∈ D} and
         j ∈ O at random
6:       SD_k.distv ← SDistV(S, SD_k)
7:    end for
8:    generate SD'_1, ..., SD'_{m_max} by sorting SD_1, ..., SD_{m_max} in
      the increasing order of SD_k.distv (1 ≤ k ≤ m_max)
9:    k ← 1, UpDated ← false
10:   while k ≤ m_max and UpDated = false do
11:      S' ← RandomSelectionInSubSubDomain(SD'_k, n)
12:      if Sat(S') > Sat(S) then
13:         S ← S', UpDated ← true
14:      else
15:         k ← k + 1
16:      end if
17:   end while
18:   if k = m_max + 1 then SearchDone ← true, and give up
      the subdomain search
19: end while
20: return S
```

Algorithm 5 *TemporarySolution Function*

```
1: δ ← Maxloc, S^cm ← ∅
2: while |S| < n do
3:    select v ∈ {v'|⟨v', *, *, *⟩ ∈ D} at random
4:    select j ∈ O at random
5:    q ← argmax_{q∈Q_v} PerCostSat(q)
6:    S ← S ∪ {⟨v, j, q, *⟩}
7:    V' ← V' − {v}, O ← O − {j}
8: end while
9: S^cm ← Modify(S)
10: S^cm ← LocalSearch(S, S^cm, δ)
11: return S^cm
```

以下に擬似コードを用いて詳細に述べる．まず観光スケジュール S，観光順 O，およびフラグ $SearchDone$ を初期化する．2 行目では，関数 $TemporarySolution()$ を用いて，暫定解 S を生成する．関数 $TemporarySolution()$ の擬似コードを Algorithm5 に示す．関数 $TemporarySolution()$ は暫定解算出フェイズとほぼ同様の処理を行っているので，動作の説明は Algorithm2 の説明を参照されたい．ここでは，予備実験により，$Maxloc = 3 \cdot |V|$ の値を用いている．4–8 行目では，部分解空間 D の中に，複数の部分部分解空間 $SD_1, ..., SD_{m_{max}}$ を生成する．つまり，同じ観光候補地を巡回する中で，巡回順序を m_{max} 種類だけ設定する．γ は部分部分解空間の生成個数を示すパラメータである．実験では，文献 [10] と [17] および予備実験により，γ は n とする．また，6 行目で，S に属する観光順空間と SD_k の間の距離を関数 $SDistV()$ で計測する．たとえば，関数 $TemporarySolution()$ を用いて，暫定

解 S= $\{\langle v_0,0,*,*\rangle\ \langle v_1,1,q_2,0\rangle\ \langle v_2,2,q_1,0\rangle\ \langle v_3,3,q_3,10\rangle\ \langle v_4,4,q_4,5\rangle\}$ を生成し，$m_{max}=4$ 個の部分部分解空間 $SD_1=\{\langle v_0,0,*,*\rangle\ \langle v_1,2,*,*\rangle\ \langle v_2,1,*,*\rangle\ \langle v_3,4,*,*\rangle\ \langle v_4,3,*,*\rangle\},...,SD_4=\{\langle v_0,0,*,*\rangle\ \langle v_1,2,*,*\rangle\ \langle v_2,1,*,*\rangle\ \langle v_3,3,*,*\rangle\ \langle v_4,4,*,*\rangle\}$ をランダムに作成したとする．それぞれの部分部分解空間と，S に属する観光順空間との間の距離は $SD_1.distv=3,...,SD_4.distv=2$ である．関数 $SDistV()$ の説明は後述する．8 行目で，計測された距離に応じ，距離の近いものから部分部分解空間をソートする．次に，9–17 行目では，これら部分部分解空間内 SD'_k を順に，関数 $RandomSelectionInSubSubDomain()$ を呼び出して探索する．得られた解 S' が改善された場合には，暫定解 S を更新する．そして 4 行目に戻って全部分部分解空間を再生成し，更新された暫定解 S に近い部分部分解空間から再度探索を開始する．最後の部分部分解空間まで探索しても改善されなければ，フラグ $SearchDone$ は $true$ に設定し，S が Algorithm4 により得られた解となる．

3.2.2.2 *DistV* と *SDistV* 関数

提案手法では，部分解空間や部分部分解空間を解空間の近さによりソートする．ここでは，この近さを与える 2 つの関数を定義する．

(1) 関数 $DistV$ は以下の式で定義される．

$$DistV = n - v_{same} \tag{9}$$

ここでは，v_{same} は 2 つのスケジュールの内の同じ観光候補地の数であり，n は経路の目的地の数である．たとえば，2 つのスケジュール $S1=\{d_0.v_0,d_1.v_1,d_2.v_2,\ d_3.v_3,d_4.v_4,d_5.v_5\}$，$S2=\{d_0.v_0,d_1.v_3,d_2.v_4,d_3.v_5,d_4.v_6,d_5.v_7\}$ の間の近さを計算する．同じ観光候補地は v_0，v_3，v_4，v_5 なので，$v_{same}=4$，$DistV(S1,S2)=6-4=2$ となる．

(2) 関数 $SDistV$ は以下の式で定義される．

$$SDistV = n - j_{same} \tag{10}$$

ここでは，j_{same} は 2 つのスケジュールの内に存在する同じ “辺” つまり目的地間の隣接の数である．また，n はスケジュールの “辺” の数である．以下の 2 つのスケジュール $S1'$ と $S2'$ では，目的地間の “辺” は “–” で表示する．たとえば，2 つのスケジュール $S1'=\{d_0.v_0\text{–}d_1.v_1\text{–}d_2.v_2\text{–}d_3.v_3\text{–}d_4.v_4\text{–}d_5.v_5\text{–}d_0.v_0\}$，$S2'=\{d_0.v_0\text{–}d_1.v_2\text{–}d_2.v_1\text{–}d_3.v_3\text{–}d_4.v_5\text{–}d_5.v_4\text{–}d_0.v_0\}$ の間の近さを計算する．同じ辺は $d_1.v_1$–$d_2.v_2$, $d_4.v_4$–$d_5.v_5$（ここでは，前後が逆でも同一の “辺” であると見なす）なので，$j_{same}=2$，$SDistV(S1',S2')=6-2=4$ となる．

3.2.2.3 *RandomSelectionInSubSubDomain* 関数

関数 $RandomSelectionInSubSubDomain(SD,n)$ は，Algorithm4 で生成した部分部分解空間 SD 内において観光方式をランダムに決定したスケジュールを返す関数である．この関数の擬似コードを Algorithm6 に示す．2–6 行目では，部分部分解空間 SD 内において，各目的地の観光方式をランダムに選ぶ．7 行目で，観光候補地，観光順番と観光方式を決定したスケジュールに対し，関数 $Modify()$ を適用し，適切に休憩を入れた観光スケジュールの満足度を算出する．

Algorithm 6 *RandomSelectionInSubSubDomaIn Function*

```
1: S ← ∅
2: while |S| < n do
3:     select d ∈ {d'|⟨v', j', *, *⟩ ∈ SD} at random
4:     select q ∈ Q_{v'} at random
5:     S ← S ∪ {⟨v, j, q, *⟩}
6: end while
7: S ← Modify(S)
8: return S
```

提案アルゴリズムは発見的アルゴリズムであり，前回のループで発見されたものよりも改善された解が見つかる限りアルゴリズムは終了しない．そのため，新しい解が次々と発見された場合，ループはそのつどリセットされ，提案手法のワーストケースとなり，時間計算量は $O(n!)$ になる．

4. 評価実験

提案手法の性能を確認するために，Java 言語でシミュレータを作成し，コンピュータシミュレーション実験を行った．シミュレーションを実行した PC は CPU が Celeron 2 GHz，メインメモリが 2 GB，OS が Windows XP である．提案手法の性能を確認するにあたり，解の最適性および演算時間それぞれの観点において，提案手法と複数ある参考手法（4.1 節）と比較し，提案手法の性質を考察する（4.2 節）．

(1) 解の最適性：スケジュールの満足度を指標とし，異なるサイズの候補地集合において，全探索を含む複数の参考手法と比較し，各種手法で出力したスケジュールの満足度がどれほど最適値に近づけるかを考察した．

(2) 演算時間：各種手法が解を求めるまでの演算時間を指標とし，良い解を実用時間内に求められるかについて考察した．

観光者の体力設定として，最大体力値 $MaxSt$ を 900 とし，休憩状態において毎分体力回復率 $habitus=10$ と設定した．実験では，13,500 m × 9,000 m のフィールドに観光候補地をランダムに含んだインスタンスを計 10 個を用いた：

- インスタンス [小]：マップ id1～5，含まれる観光候補地の数は 10.
- インスタンス [大]：マップ id6～10，含まれる観光候補地の数 20.

各候補地の重要度は 1–100 の値をランダムで設定した．

表 3 観光候補地の例
Table 3 The data structure of the sightseeing spot.

観光方式	滞在時間	満足度	体力消費
山を登る a	1 h	27	410
山を登る b	1.5 h	72	615
山を登る c	2 h	85	819
ジョギングと歩行の組合せ a	1 h	48	341
ジョギングと歩行の組合せ b	1.5 h	79	512
ジョギングと歩行の組合せ c	2 h	75	682

実験を簡単化するため，各候補地の観光方式を 2 種類とした．また各観光方式の滞在時間は，1 時間，1.5 時間および 2 時間とした．**表 3** はこれらの設定に基づく候補地例の 1 つである．候補地の観光方式は計 6 通りあり，観光方式ごとの消耗体力値は統計情報 [18] に基づいて設定した．また，移動方式を自動車と想定し，すべての実験において，目的地間の移動速度を 30 km/h とした．なお，移動距離に対する制御パラメータ α を 3 とし，移動距離が 1 km になるごとに満足度は 3 点減点される．

4.1 参考手法

提案手法の性能を評価するため，複数の参考手法と比較実験を行った．

- **欲張り法**：出発地から移動した後，次の目的地・観光方式の満足度を最大化することだけを考えて選択する手法．観光者の体力がなくなると，休憩を入れて，帰着時刻まで目的地を巡回する．この手法は次の目的地の満足度だけに注目するため，移動距離の総和を最短化しようとしない．
- **GA 法**：複数日にわたる観光のためのスケジュール作成手法 [7] を利用した手法．元の手法は体力を考慮していなかったが，提案手法と同様に体力を考慮して休憩を挿入するよう修正している．
- **PS 法**：暫定解算出フェイズを用いず，ランダムに生成された解から近傍探索フェイズのみで改善する手法．従来の捕食法（Predatory Search, PS）に相当する．
- **TSS 法**：提案手法の暫定解算出フェイズのみを用いた方法．TSS とは *Temporary Solution Search* の略記である．TSS 法を実装する際，妥当な欲張り関数が必要である．複数の欲張り関数候補を予備実験（詳細実験結果を割愛）で求めた結果，3.2.1.1 であげた関数 *PerCostSat*() を採用している．
- **全探索法**：すべての組合せを全探索により求め，最大の満足度のスケジュールを得る方法．

4.2 シミュレーション実験

4.2.1 解の満足度

算出スケジュールの満足度を評価するために，5 種類の参考手法との比較実験を行った．また，大小両方の観光地

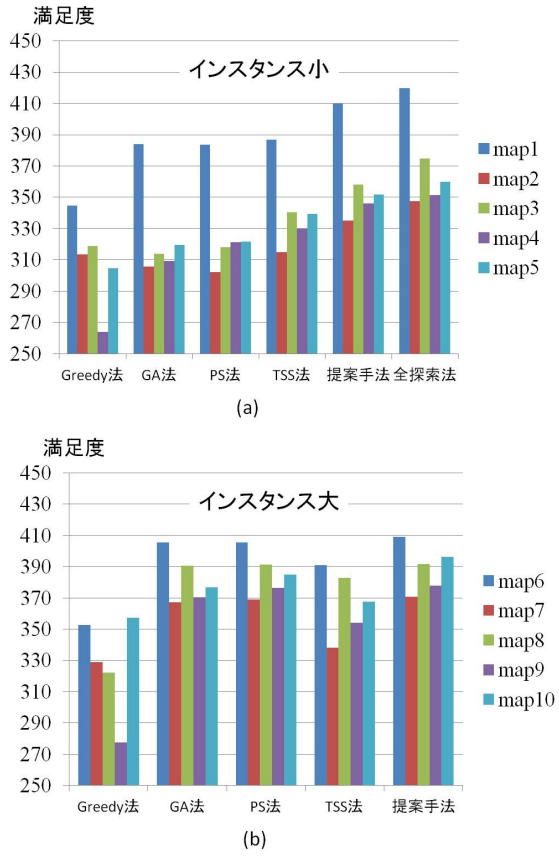

図 3 満足度の比較結果
Fig. 3 Results of comparison for satisfaction degree.

集合（インスタンス）を実験に用いることで，提案手法のスケーラビリティを確認した．それぞれの実験結果は 30 試行の平均となっている．また，GA 法と PS 法に対し，探索時間を制限し，演算時間が同じ条件のもとで比較を行った．

インスタンス [小] の結果を**図 3** (a) と**表 4** に示す．全探索法では，最適解を求めるのに 109.75 分かかったのに対し，提案手法は約 13 秒で最適解の 95.65%以上の満足度を持つ解を求めることができた．GA 法と PS 法では，提案手法よりも長い演算時間を設定したにもかかわらず，得られた満足度は提案手法より低い．一方，Greedy 法と TSS 法は短い時間で演算を終えるが，解の質は低く，局所解に陥っていると考えられる．欲張り法と比較したところ，提案手法は，1.36 倍の満足度を持つ解を得ることができた．

インスタンス [大] の結果は図 3 (b) と表 4 に示されている．全探索法で最適解を求めることは実用時間内で不可能なため，その結果を掲示していない．提案手法と他の手法を比較した結果，ほぼ同様の満足度の解を得るまでに，提案手法では 93.75 秒，GA 法と PS 法で 160 秒程度かかった．提案手法は解の質を維持しつつ，GA 法と PS 法より明らかに計算量が少ないことが分かる．これと比べ，TSS 法と欲張り法はインスタンス [小] の場合と同様な傾向が見られ，短い演算時間で演算を求められるが，解の質が低い．

表 4 平均計算時間の比較結果
Table 4 Results of comparison for computing time.

平均計算時間	欲張り法	GA 法	PS 法	TSS 法	提案手法	全探索法
インスタンス小	0.6 ms	19.43 s	20.16 s	0.52 s	13.68 s	109.75 min
インスタンス大	10.5 ms	164.21 s	167.45 s	6.28 s	93.75 s	–

表 5 全探索法の計算時間評価
Table 5 Computing time of Brute-force search.

インスタンス	候補地数（6）	候補地数（8）	候補地数（10）	候補地数（12）
全探索法	1.37 min	18.44 min	106.2 min	424.93 min

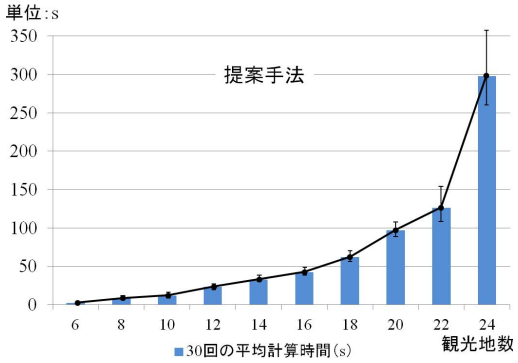

図 4 提案手法の計算時間評価
Fig. 4 Computing time of proposed method.

4.2.2 計算時間の評価

提案手法の計算時間を評価するため，全探索の計算時間と比較した．実験には，ランダムに生成されたインスタンス 10 個を用いた．10 個のインスタンスに含まれる観光地候補数は 6–24 個（6，8，10，12，14，16，18，20，22，24）である．また，各候補地の観光方式の満足度もランダムに生成した．全探索は探索候補数が増えるにつれ，計算時間が爆発的に増加するため，観光候補地数（6，8，10，12）の場合についてのみ計算時間を計測した．実験結果を**表 5** に示す．候補地数 12 の場合，計算時間が 7.08 時間かかることが分かった．**図 4** に，提案手法の計算時間（30 試行の平均）を示す．候補地数 18 以下の場合，1 分以内で解を得ることができていることが分かる．提案手法の計算時間は候補地数が 20 まで，緩やかに伸びている．各試行における計算時間のばらつきも大きくない．しかし，提案手法は候補地数が 22 のとき，計算時間自体はさほど長くなっていないが，各試行の計算時間のばらつきが大きくなり，候補地数が 24 のとき，計算時間が 300 秒近くまでふくれあがった一方，計算時間のばらつきもさらに顕著になった．提案アルゴリズムでは，時間制約内の巡回する目的地と観光方式をランダムに決定するので，候補地数が多いほど，経路探索時の運の影響が大きくなる．提案手法は最悪の場合の時間計算量は $O(n!)$ であるものの，各目的地とその観光方式のセットをあらかじめ精査するなどの工夫により，解空間を著しく縮小することが可能と思われ，平均計算量は大幅に削減されることが期待できる．平均計算量の正確な見積りは今後の課題である．

5. まとめ

本研究では，観光者の体力と観光時間を制約とし，合計の満足度を最大化するような観光スケジュールの立案手法を提案した．提案手法は，TSP の有効解法の 1 つである捕食法を利用している点に特色がある．解の探索効率を評価するため，提案手法と全探索で求めた解と比較した．その結果，最適解の 95.65%以上の満足度を持つ解を平均 12.99 秒で得ることができた．また，欲張り法と比較したところ，20 個の目的地数を持つインスタンスにおいて 1.36 倍の満足度を持つ解を得ることができた．

今後の課題として，移動中の体力を考慮することがあげられる．厳しい天候の下では，移動経路が屋外か屋内か，またバスなどを利用できるかが体力消耗に大きな影響を与えることが考えられ，これらの利用スケジュールをうまく立案することは重要である考えられる．また，目的地の観光自体が体力の回復につながる場合が考えられる．たとえばレストランで食事をしたり，温泉で休憩したりするなど，目的地での休憩も考慮したスケジューリング手法を提案する予定である．

参考文献

[1] Baus, J., Krüger, A. and Wahlster, W.: A Resource Adaptive Mobile Navigation System, *Proc. 2002 Int'l Conf. Intelligent User Interfaces 2002* (*IUI-02*), pp.15–22 (2002).
[2] Tang, F., You, I., Guo, M. and Guo, S.: Context-aware workflow management forintelligent navigation applications in pervasive environments, *Intelligent Automation and Soft Computing*, Vol.16, No.4, pp.605–619 (2010).
[3] Butz, A., Baus, J., Krüger, A. and Lohse, M.: A Hybrid Indoor Navigation System, *Proc. 2001 Int'l Conf. Intelligent User Interfaces 2001* (*IUI2001*), pp.25–33 (2001).
[4] Cheverst, K., Davies, N., Mitchell, K., Friday, A. and Efstratiou, C.: Developing a context-aware electronic tourist guide: some issues and experiences, *Proc. 2000 ACM Special Interest Group on Computer-Human Interaction* (*SIGCHI-00*) pp.17–24 (2000).
[5] Rehrl, K., Leitinger, S., Bruntsch, S. and Mentz, H.J.:

Assisting orientation and guidance for multimodal travelers in situations of modal change, *Proc. 2005 IEEE Int'l Conf. Intelligent Transportation Systems* (*ITSC-05*) pp.407–412 (2005).
[6] Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, *European Journal of Operational Research*, Vol.126, No.1, pp.106–130 (2000).
[7] 木下隆正，永田宗伸，村田佳洋，柴田直樹，安本慶一，伊藤　実：複数日にわたる観光のためのパーソナルナビゲーションシステム，情報処理学会論文誌，Vol.47, No.12, pp.3179–3187 (2006).
[8] Linhares, A.: State-space search strategies gleaned from animal behavior: A traveling salesman experiment, *Biological Cybernetics*, Vol.78, No.3, pp.167–174 (1998).
[9] Linhares, A.: Synthesizing a predatory search strategy for VLSI layouts, *IEEE Trans. Evolutionary Computation*, Vol.3, No.2, pp.147–152 (1999).
[10] Liu, C. and Wang, D.: Predatory search algorithm with restriction of solution distance, *Biological Cybernetics*, Vol.92, No.5, pp.293–302 (2005).
[11] Ahuja, R.K., Ergun, O., Orlin, J.B. and Punnen, A.P.: A survey of very large-scale neighborhood search techniques, *Discrete Applied Mathematics*, Vol.123, pp.75–102 (2002).
[12] Iordache, S.: Consultant-Guided Search – A New Metaheuristic for Combinatorial Optimization Problems, *Proc. 2010 Genetic and Evolutionary Computation* (*GECCO'10*), ACM Press (2010).
[13] Petrica, C.P. and Iordache, S.: A hybrid heuristic approach for solving the generalized traveling salesman problem, *Proc. 2011 Genetic and Evolutionary Computation* (*GECCO'11*), pp.481–488 (2011).
[14] Maruyama, A., Shibata, N., Murata, Y., Yasumoto, K. and Ito, M.: P-Tour: A Personal Navigation System for Tourism, *Proc. World Congress on ITS*, pp.18–21 (2004).
[15] 武　兵，村田佳洋，柴田直樹，安本慶一，伊藤　実：天気変化を考慮した観光スケジュール群の探索アルゴリズム，情報処理学会論文誌数理モデル化と応用，Vol.3, No.1, pp.87–97 (2010).
[16] Behzad, A. and Modarres, M.: A new efficient transformaiton of the generalized traveling salesman problem into traveling salesman problem, *Proc. 22rd International Conference on Software Engineering* (*ICSE 2002*), pp.43–46 (2002).
[17] Kenneth, D.B., Andrew, B.K. and Sudhakar, M.: A new adaptive multi-start technique for combinatorial global optimizations, *Operations Research Letters*, Vol.16, No.2, pp.101–113 (1994).
[18] Club.Panasonic：運動・生活活動の消費カロリー，入手先 ⟨http://club.panasonic.jp/diet/exercise/mets/index.html⟩.

武 兵

2004 年中国山東理工大学計算機学部卒業．2010 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．同年 4 月より現在，同大学情報科学研究科博士後期課程に在学中．観光用パーソナルナビケーションシステムに関する研究に従事．

孫 為華　（正会員）

2003 年，2005 年，2008 年にそれぞれ大阪大学基礎工学部卒業，同大学大学院情報科学研究科博士前期課程修了，同大学院情報科学研究科博士後期課程修了．2008 年より，奈良先端科学技術大学院大学情報科学研究科助教．博士（情報科学）．モバイルアドホック，車車間通信に関する研究に従事．IEEE 会員．

村田 佳洋　（正会員）

2003 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年奈良先端科学技術大学院大学情報科学研究科助手．現在，広島市立大学情報科学研究科准教授．遺伝的アルゴリズム，エージェント技術等の研究に従事．

安本 慶一　（正会員）

1991 年大阪大学基礎工学部情報工学科卒業．1995 年同大学大学院博士後期課程退学後，滋賀大学経済学部助手．2002 年奈良先端科学技術大学院大学情報科学研究科助教授，2011 年より同研究科教授．博士（工学）．モバイルコンピューティング，ユビキタスコンピューティングに関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．

伊藤 実 （正会員）

1977年大阪大学基礎工学部卒業，1979年同大学大学院基礎工学研究科博士前期課程修了．1979年より大阪大学基礎工学部助手．1986年より大阪大学基礎工学部講師．1989年より大阪大学基礎工学部助教授．1993年より奈良先端科学技術大学院大学情報科学研究科教授．現在に至る．工学博士．データベース理論，効率的なアルゴリズム開発等の研究に従事．ACM，IEEE，電子情報通信学会各会員．